Décio Cristóvão Andrade Miranda

Negociação em Bolsa em Dispositivos Móveis

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
Setembro de 2012

Décio Cristóvão Andrade Miranda

Negociação em Bolsa em Dispositivos Móveis

Dissertação submetida à Faculdade de Ciências da
Universidade do Porto como parte dos requisitos para a obtenção do grau de
Mestre em Engenharia de Redes e Sistemas Informáticos

Orientador: Dr. Bruno Alexandre Carvalho Rodrigues Oliveira
Co-orientador: Professor Dr. António Mário da Silva Marcos Florido

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
Setembro de 2012

Dedico este trabalho, desenvolvido ao longo de nove meses de estágio curricular, à minha família, principalmente aos meus pais que tanto lutaram para que este meu sonho se tornasse realidade. Aos meus irmãos, pela força, amizade, companheirismo e confiança. À minha namorada, pelo amor e carinho, apoio incondicional e paciência.

Agradecimentos

O meu muito obrigado às minhas avós, por me apoiarem à distância.

Aos meus sogros, que sempre estiveram presentes quando precisei.

Aos meus amigos, pelo estímulo e companhia.

Ao meu orientador de estágio, Dr. Bruno Oliveira, pela paciência, dedicação e amizade.

Ao meu co-orientador e professor, Dr. Mário Florido, pela preciosa orientação e compreensão.

Resumo

Este projeto surge integrado no estágio curricular, ao abrigo da Faculdade de Ciências da Universidade do Porto, a fim de concluir o curso de Mestrado Integrado em Engenharia de Redes e Sistemas Informáticos. "Negociar em Bolsa em Dispositivos Móveis" foi o projeto que suscitou em mim um sentimento de curiosidade e interesse, principalmente por duas expressões, "Negociar" e "Dispositivos Móveis", no conjunto de muitas propostas de estágio recebidas pela Faculdade de Ciências.

A palavra "Negociar", que se refere à grandeza e complexidade da negociação em bolsa, era-me desconhecida e foi encarada como um desafio enriquecedor da minha futura atividade profissional.

A expressão "Dispositivos Móveis", pelo surgimento revolucionário dos *smartphones* que estão à mercê de qualquer pessoa e que são máquinas cada vez mais pequenas e potentes.

Portanto, este projeto reflete o trabalho de implementação de uma aplicação móvel de negociação em bolsa, para *iPhone* e *iPad*, que surge como parte integrante e consumidora dos sistemas de suporte a funções de *Backoffice* e *Middleoffice* de negociação em bolsa da empresa Finantech - Sistemas de Informação, Lda.

Ao longo deste documento serão analisadas algumas aplicações móveis de negociação em bolsa, como termo de comparação, dando a conhecer um pouco do estado da arte deste projeto. Serão apresentados os objetivos, as funcionalidades e as características da aplicação e a sua interação com o utilizador. Será estudada ao pormenor toda a arquitetura da aplicação, desde os seus *data handlers* aos seus *listeners* consumidores dos dados, interfaces de ligação ao servidor e toda a lógica estrutural e de comunicação com o servidor.

E por último serão descritas as tecnologias e ferramentas utilizadas ao longo do desenvolvimento da aplicação.

Abstract

This project is integrated in the curricular internship, under the Faculty of Sciences of University of Porto, to complete the Master in Network Engineering and Systems. "Trading in Mobile Devices" was the project that aroused in me a very strong feeling of curiosity and interest, mainly due to the expressions "Trading" and "Mobile Devices", amidst the many internships proposals that the Faculty of Sciences offered.

The word "Trading", which of course refers to the trading in the stock market, was unknown to me and I looked at it has a very enticing challenge to my future professional activity.

The term "Mobile Devices", the emergence of revolutionary Smartphones that are at the mercy of anyone and in this day and age are "growin" amazingly small and powerful.

Therefore, this project reflects the work of developing a mobile application of trade in the Stock Market, for iPhone and iPad, which comes as part of the infrastructure of the company Finantech - Sistemas de Informação, Lda that allowing the operations in the system.

Along this thesis Will be presented the objectives, functionalities and features of the application and its interaction with the user. Will be studied in detail the entire architecture of the application, their data handlers, their listeners data consumers, interfaces connecting to the server and all the structural logic and communication with the server.

For last will be described the technologies and tools used throughout the application development.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

A negociação eletrónica, muito devido à melhoria das tecnologias de informação no final do séc. XX, tornou a necessidade de um local físico menos importante. Os *traders*[1] podiam negociar remotamente de uma forma clara, eficiente e rápida. Formam-se salas virtuais de negociação com compradores e vendedores. O número de investidores aumentou até então, tal como a liquidez de ações transacionadas em relação ao antigo método de negociação, uma vez que este tipo de negociação é independente da localização.

Mais recentemente, a negociação em bolsa tem passado, de certa forma, dos grandes ecrãs para os pequenos ecrãs. Com o aparecimento e crescimento exponencial da tecnologia móvel, que é bem caraterizada pelo aparecimento dos sistemas operativos *Android* da *Google* e *iOS* da *Apple*, tornou-se mais fácil e possível o desenvolvimento de aplicações móveis capazes de fornecer qualquer tipo de informação, ação ou serviço de bolsa disponibilizado pelos demais métodos de negociação.

A negociação móvel usufrui da tecnologia *wireless* que permite ao investidor, seja *trader* ou não, negociar a partir do seu *smartphone* ou *tablet*, esteja onde estiver, a qualquer altura, assim como consultar e gerir o seu portfólio e conta.

[1] Pessoa ou entidade que vende e compra instrumentos financeiros.

1.1 O Projeto

1.1.1 Objetivos

O projeto, *"Negociação em Bolsa em Dispositivos Móveis"*, como a própria descrição indica, tem como objetivo o desenvolvimento de um aplicação de *trading*, isto é, do processo de negociação de instrumentos financeiros, denominada SifoxDeal Mobile, para dispositivos móveis, nomeadamente para *iPhone* e *iPad*, equivalente às restantes aplicações já desenvolvidas pela Finantech para ambiente *desktop*. O SifoxDeal Mobile vem mudar o modo como o cliente final interage com a instituição (banco ou *broker*[1]) a que está filiado: uma interação mais eficaz e eficiente. De um modo geral, a aplicação permite a consulta de dados de mercado em tempo real e a realização de transações nos maiores mercados e *brokers* nacionais e mundiais.

Destina-se essencialmente a três tipos de clientes: o cliente particular, ao qual é fornecido o acesso a dados de mercado, a pesquisa de títulos nos dicionários, a gestão das suas ordens, carteira de títulos e contas financeiras com os respetivos movimentos; o cliente institucional, com acesso a todas as funcionalidades; e por último, o *trader* com acesso a todas as funcionalidades e uma extra face aos outros tipos de cliente, uma funcionalidade multi-cliente, que permite ao *trader* acompanhar as ordens, os negócios, a carteira e contas dos seus vários clientes.

A aplicação, desenvolvida no âmbito desta tese, deverá ainda tirar partido das infra-estruturas disponíveis na empresa, que fornecem um conjunto de funcionalidades que dão suporte a vários aspetos do desenvolvimento da aplicação, nomeadamente, a disponibilização de informação em *real time* por *HTTP* e a gestão de utilizadores/acessos.

1.1.2 Funcionalidades

Para este projeto foram delineadas algumas funcionalidades mínimas a serem desenvolvidas, tais como:

- Visualização de dados de mercado;
- Visualização de gráficos de *intraday*;

[1]Intermediário financeiro, ou seja, é aquele que executa a transação entre um comprador e um vendedor.

- Visualização do livro de ordens;
- Envio, modificação e anulação de ordens;
- Visualização da carteira e do financeiro em *real time*;
- Pesquisa de títulos nos dicionários.

1.1.3 A empresa

Este projeto foi desenvolvido na empresa de Sistemas de Informação, S.A., adiante designada por Finantech, anteriormente conhecida por Arrábida Informática, fundada em Setembro de 1994 por um grupo de antigos colaboradores da Bolsa do Porto. Esta empresa é líder em Portugal no desenvolvimento de sistemas de informação para o mercado de capitais. A sua principal atividade baseia-se no desenvolvimento e manutenção de aplicações de suporte a funções de *BackOffice*[1], *MiddleOffice*[2], *FrontOffice*[3] e negociação *online* de *brokers*. As soluções desenvolvidas são permanentemente aperfeiçoadas e, sempre que a oferta *standard* for insuficiente para a totalidade dos requisitos, a Finantech desenvolve *software* à medida do cliente. A Empresa fornece ainda um serviço de *helpDesk*[4] alargado, capaz de responder a qualquer eventual problema em tempo útil.

1.1.4 Casos Reais

Inicialmente o projeto tinha como objetivo o desenvolvimento de uma aplicação para *iPhone* com algumas funcionalidades a serem implementadas. À medida que cada funcionalidade foi sendo implementada e desenhada, foram feitas demonstrações pelos comerciais da Finantech aos clientes. As sugestões foram surgindo, quer por parte dos clientes quer por parte dos próprios colaboradores da Finantech, mais a nível do desenho da interface e navegação da aplicação. A aplicação foi evoluindo a bom ritmo, com os clientes a fornecerem um bom *feedback*.

[1]Conjunto de funcionalidades essenciais para responder às necessidades dos intermediários financeiros a nível de processamento, controlo e suporte a um conjunto de ativos financeiros.

[2]Conjunto de ferramentas essenciais para a conexão com os mercados financeiros. Garantem também uma interação entre a área BackOffice e a MiddleOffice.

[3]Conjunto de soluções necessárias para o desempenho da atividade de trading.

[4]Define o serviço de apoio ao cliente para resolução de eventuais problemas.

O interesse dos clientes pela aplicação *iPhone* fez surgir a hipótese de a desenvolver para o *iPad*. Esta permitiria aos clientes, no âmbito empresarial, como corretoras e bancos, terem uma visão mais alargada e próxima de um computador convencional, a que os *traders* estão normalmente habituados. Assim sendo, o objetivo do projeto foi alargado, passando a abranger a elaboração da mesma aplicação para o *iPad*.

Após a concretização de uma versão estável da aplicação, quer para *iPhone* quer para *iPad*, a aplicação entrou em fase de testes no Banco Invest e em produção na corretora Fincor.

1.2 Organização da tese

A tese apresentada encontra-se organizada da seguinte forma: o capítulo 2, dedicado ao estado da arte, aborda as funcionalidades das aplicações de negociação em bolsa para as plataformas *desktop* e *mobile*; o capítulo 3, dedicado à aplicação, introduz os padrões de desenho de software e toda a arquitetura da aplicação SifoxDeal Mobile; o capítulo 4, referente à utilização, descreve a navegação dos vários modos de operação da aplicação; o capítulo 5, a respeito da mobilidade, descreve a comunicação e o modelo cliente-servidor subjacente à arquitetura da aplicação; o capítulo 6, dedicado à descrição das tecnologias e ferramentas utilizadas, e, por fim, o capítulo 7, onde se conclui o trabalho realizado e aborda-se a possibilidade de trabalho futuro.

Capítulo 2

Estado da Arte

Apesar de a Bolsa não ser meramente um mercado de ações, esta é a sua vertente mais conhecida. Cada vez mais as empresas arriscam a entrada no mercado de valores por forma a obter recursos e crescerem como empresa ou sociedade. A acompanhar esta tendência, os sistemas de informação evoluíram de modo a proporcionar uma forma mais facilitada de negociar em diversos instrumentos financeiros, bem como obter uma análise detalhada do mercado atual através de ferramentas automatizadas de análise, que facilitam as tomadas de decisão ou até mesmo que efetuam operações de forma automática.

Comprar ou vender ações tornou-se hoje um processo simples, rápido e independente da localização geográfica, muito devido ao aparecimento dos *smartphones*. Portanto, a negociação em bolsa passou a marcar presença em qualquer parte do globo à distância de um *click*, respondendo às exigências dos muitos intermediários financeiros para a sua atividade de *trading*.

2.1 Plataformas

2.1.1 Desktop

As aplicações *desktop* atuais permitem a monitorização e negociação nos mercados de capitais em tempo real, em que podem ser visualizados não só *tickers*[1] de cotações

[1]Em bolsa é um termo utilizado com a finalidade de publicitar algumas informações de um título do momento.

mas também de índices e de negócios.

A informação de dados de mercado é a mais importante das informações da atividade bolsista que normalmente é mantida numa janela em forma de grelha em que cada linha corresponde a um título. A informação organizada em linhas e colunas permite a sua visualização extensa e estruturada, para que seja possível uma análise rápida e clara.

Para cada título, a maioria das aplicações permite a consulta de informações como:

- Gráfico *intraday*[1];
- Gráfico histórico;
- Gráfico de cruzamentos;
- Detalhe do título;
- Ficha do título;
- Profundidades agrupadas ou desagrupadas por preço.

O gráfico *intraday* possibilita a consulta evolutiva da cotação de um título ao longo do dia, ou seja, desde a abertura do mercado até ao seu fecho, assim como a consulta da última cotação, do valor mínimo, valor máximo, preço de abertura, preço de fecho, as quantidades transacionadas e *Volume Weighted Average Price (VWAP)* do título.

Relativamente ao gráfico histórico, este permite a análise evolutiva da cotação ou das quantidades transacionadas de um título num período de tempo, o qual é estabelecido por uma data de início e uma data de fim. O utilizador, para além de poder visualizar o registo histórico num intervalo de tempo que o próprio pode definir, possui também ao seu dispor esse registo (histórico) com datas predefinidas, como por exemplo, o histórico de uma semana, um mês, três meses, seis meses, um ano, dois anos e cinco anos.

Ambos os gráficos, *intraday* e histórico, são muitos úteis e comuns às aplicações *desktop* de negociação. A sua importância reside essencialmente na simplicidade organizacional da informação, não poupando contudo a informação essencial para uma boa observação

[1]Gráfico que permite a análise diária numa bolsa de valores com o objetivo de se observar a oscilação do preço e quantidades transacionadas de um título desde a abertura do mercado até ao momento.

sobre a evolução dos preços. Permitem ao utilizador uma análise técnica que facilmente leva à deteção de subidas e descidas de preços, ao conhecimento da tendência principal dos movimentos dos mesmos e, até mesmo, das pequenas oscilações de preço que vão sucedendo. Esta análise técnica é uma ajuda relevante aquando da tomada de decisões, designadamente a compra e venda de ativos.

Por sua vez, o gráfico de cruzamentos permite a comparação de variações percentuais ao longo do tempo entre vários gráficos de cotações de títulos diferentes. Esta funcionalidade é útil, por exemplo, na análise entre dois títulos de mercado que o utilizador possui em carteira e que está na indecisão de comprar ou vender um dos dois. Portanto, tal como nos gráficos anteriores, este auxilia o utilizador na tomada de decisões.

O detalhe do título apresenta toda a informação referente ao mesmo, como por exemplo, o preço atual, o preço do comprador, o preço do vendedor, preço máximo e mínimo, variação absoluta e percentual, preço de abertura e fecho, volume de ações, hora do último negócio e quantidades transacionadas, entre outras. A ficha, por sua vez, fornece informações relativas às condições e regras do mercado onde está inserido o título em questão.

Este tipo de *software* também possui toda a informação e funcionalidades referente às ordens. Tipicamente é fornecido ao utilizador uma lista das ordens de compra e venda ativas para uma ação num dado momento, para que seja possível, por exemplo, verificar se existe uma forte pressão de venda, ou se, pelo contrário, existem mais compradores do que vendedores e possa, se assim o desejar, agir em conformidade. Após o envio de uma ordem com sucesso, seja de compra ou de venda, automaticamente é acrescentada à grelha uma nova linha contendo a informação descritiva da ordem sobre a qual o utilizador pode efetuar algumas operações como a anulação, duplicação, modificação, rejeição, desbloqueio, acompanhamento e consulta dos seus eventos e execuções.

A respeito das profundidades agrupadas ou desagrupadas por preço, a profundidade mais usada, nomeadamente em Portugal, é a "profundidade 5", ou seja, são agrupadas pelo preço as 5 ordens de compra com preço mais alto e as 5 ordens de venda com o preço mais baixo. As quantidades das ordens são somadas, o que dá ao utilizador uma noção da tal pressão de venda ou compra já falada anteriormente. Talvez por se tratar basicamente de um leilão, a uma ordem de compra dá-se o nome de *Bid* (melhor oferta de compra) e a uma ordem de venda o nome de *Ask* (melhor oferta de venda).

Algumas aplicações implementam funcionalidades como a negociação rápida, por exemplo, que pela configuração de valores por defeito, definidos pelo utilizador, permitem

negociar com apenas um *click*. Outra funcionalidade é a negociação avançada em que o utilizador pode personalizar as caraterísticas da ordem em questão, e depois proceder à sua gestão para que seja executada com sucesso. É uma ordem normalmente de grande quantidade cujo objetivo é atingir essa mesma quantidade até ao fecho do mercado, ou seja, a ordem é dividida em blocos e mandada para o mercado para ser executada pouco a pouco. Um motivo para usar este tipo de funcionalidade, poderá ser o facto de se querer adquirir (ou vender) um elevado número de ações, o que poderia ter um impacto significativo na cotação do título se fosse executada de uma só vez.

Existe também uma funcionalidade de realização de negócios, em que o utilizador pode definir o vendedor e o comprador, enviando para o sistema duas ordens distintas: uma de compra e outra de venda. Ainda a respeito das ordens, são realizadas uma série de validações aquando do seu envio, tais como o valor do saldo contabilístico, quantidade de ações disponíveis referentes ao título, entre outras.

Uma outra janela importante, normalmente implementada pelas aplicações de negociação em bolsa para *desktop*, é a janela que contém todos os eventos pelos quais todas as ordens, enviadas pelos *traders* selecionados, passaram. Esta janela não é mais do que uma tabela que contém um conjunto de *tickers* de ordens que vai sendo atualizado ao longo do tempo com mensagens de estado provenientes do sistema de negociação.

O relatório e o histórico de eventos são outras funcionalidades importantes implementadas por algumas aplicações. A maioria mantém um relatório de execuções com informações agrupadas por cliente e um histórico das ordens executadas, que permite ao utilizador observar em que condições foram executadas todas as ordens enviadas para o sistema de negociação.

A variedade de aplicações de negociação para *desktop* é imensa, tal como os tipos de utilizadores que delas usufruem, desde o utilizador doméstico com uma conta registada numa corretora ou banco com o objetivo principal de negociar, manter e gerir a sua carteira de títulos e financeiro, até ao convencional *trader* inserido numa corretora que pode ter vários clientes associados, podendo negociar com qualquer um deles, visualizar os seus negócios, a sua carteira e financeiro. A carteira consiste numa lista de todos os títulos que o cliente possui, assim como informações relevantes sobre a mesma, como valorização no mercado, cotação, valor de aquisição, etc. Em relação ao financeiro, este consiste simplesmente em informação referente às contas bancárias dos clientes, como movimentos e saldos.

2.1.2 Mobile

Esta subsecção aborda algumas aplicações móveis de negociação em bolsa no mercado. Ao contrário da anterior, que descreve as funcionalidades principais de um sistema de negociação para plataformas *desktop*, esta descreve as funcionalidades de cada aplicação móvel em específico, a fim de permitir uma análise e comparação crítica com o SifoxDeal Mobile para *iPhone* e *iPad*.

2.1.2.1 Bloomberg By Bloomberg Finance LP

A Bloomberg é uma empresa norte-americana fundada por Michael Bloomberg[6] com a ajuda de Thomas Secunda, Duncan MacMillan e Charles Zega em 1982. É uma das maiores agências de notícias para o mercado financeiro que se encontra presente praticamente na totalidade dos bancos e corretoras do mundo. Também fornece ferramentas e *software* financeiro como análise, plataformas de negociação e serviço de dados, tudo através de um terminal Bloomberg [1] [16].

A empresa cresceu para incluir um serviço global de notícias, incluindo televisão, rádio, Internet e publicação de revistas e jornais. A componente móvel foi outra novidade encontrada no percurso de crescimento recente da Bloomberg. A figura 2.1 apresenta algumas imagens da aplicação para *iPhone*:

Figura 2.1: Aplicação para iPhone da Bloomberg.

A aplicação da Bloomberg para *iPhone* é em parte semelhante ao SifoxDeal Mobile referente à disponibilização de informação de dados de mercado, gestão de carteira,

acesso às contas bancárias dos clientes e acesso a uma componente de gráficos e tabelas que ilustram, de certa forma, o impacto e a tendência dos principais negócios. Contudo, existe uma diferença de certa forma evidente, a aplicação da Bloomberg é uma aplicação direcionada à informação e propagação da mesma, ao contrário do SifoxDeal Mobile que foi desenhado e orientado para o mundo do negócio, envio e tratamento de ordens.

A Bloomberg fornece uma base alargada de categorias de notícias, como por exemplo, Economia, Tecnologia, Saúde, Energia, Títulos, Mercados emergentes, Política, Desporto, *Stocks* entre outras. Perante estas categorias, um utilizador pode criar a sua própria categoria de notícias personalizada por região, indústria ou popularidade, com notícias de títulos específicos com a possibilidade de visualização do seu preço de abertura, preço de fecho e preço máximo e mínimo durante os últimos cinco anos.

Como já mencionado, essa aplicação também fornece o acesso à informação de dados de mercado através de índices de ações, *commodities*[1], obrigações, divisas, futuros e gráficos estatísticos interativos.

A aplicação da Bloomberg possui um modo *plus* pago que se estende à condição grátis em algumas funcionalidades, como a *Bloomberg Television Live* que funciona como uma rede de notícias vinte e quatro horas, apoiada por cento e quarente e seis agências de notícias em setenta e dois países. Outra funcionalidade é a lista de *tops* que a aplicação mantém. *Top* dos vídeos que relatam as maiores histórias do mundo do negócio, vídeos relacionados com artigos e *top* entrevistas em formato áudio. A aplicação apresenta ainda a capacidade ou funcionalidade de partilhar histórias pelo *Facebook* e *Twitter* [17].

2.1.2.2 GoBulling Pro Mobile

A GoBulling é a marca do Banco Carregosa referente à negociação *online* que permite o acesso ao mercado de capitais. Fundado em 1833 e lançada em 2007 no Porto é especializada em ativos e serviços financeiros. Foi o primeiro *broker* a praticar comissão de corretagem zero em todos os mercados *Euronext* [9].

A GoBulling disponibiliza plataformas que permitem o investimento no mercado de capitais com objetivos e funcionalidades diferentes, como a plataforma GoBulling Pro, GoBulling Pro Web e GoBulling Pro Mobile.

[1]Define bens para os quais existe procura independentemente da qualidade e marca do produto no conjunto dos mercados.

A aplicação GoBulling Pro Mobile é uma aplicação com funcionalidades e características diferentes do SifoxDeal Mobile. Em comum, existe apenas o acesso à informação de dados de mercado, a gestão de ordens e a visualização de gráficos e notícias. A informação pública sobre as suas funcionalidades e caraterísticas é muito reduzida, uma vez que se trata de uma aplicação direcionada aos seus clientes diretos. A figura 2.2 apresenta imagens da aplicação para *iPhone*:

Figura 2.2: Aplicação para iPhone da GoBulling.

O utilizador, a partir dessa aplicação, tem acesso ao resumo da sua conta, que contém informações como saldo da conta, valor da conta, margem disponível e utilização da mesma. Permite a pesquisa de instrumentos financeiros para posteriormente abrir posições, colocar ordens, visualizar o respetivo gráfico e adicionar alertas de preço [10] [11].

2.1.2.3 MBolsa

MBolsa é uma aplicação desenvolvida e lançada pelo banco português Millennium bcp para dispositivos móveis com o objetivo principal de fornecer aos seus clientes o acesso à carteira de títulos e a possibilidade de negociar no mercado nacional. O bcp é o maior banco privado português fundado em 1985 [3]. Apresenta-se em Portugal sobre as marcas *Millennium bcp*, *Banque bcp* e *ActiveBank*. A figura 2.3 apresenta imagens da aplicação para *iPhone*.

A partir desta aplicação, e como cliente Millennium bcp, é possível consultar os principais índices acionistas e ações das bolsas mundiais, assim como os componentes dos índices, as mais transacionadas, as maiores subidas e descidas. Uma das funciona-

Figura 2.3: Aplicação MBolsa do banco Millennium bcp para iPhone.

lidades mais importantes desta aplicação é a consulta detalhada dos títulos em carteira e a compra e venda de ações nacionais (PSI20 e *Euronext Lisbon*) com a possibilidade de anulação de uma ordem. Fornece também o acesso ao estado das ordens de bolsa e uma componente gráfica simples ilustrativa da evolução diária (*intraday*) e histórica (1 semana, 1 e 3 meses, 1 e 5 anos) de um título ou índice.

A aplicação também está disponível para qualquer utilizador mesmo que não seja cliente, mas com restrições, apenas podem visualizar os principais índices e ações das bolsas mundiais, assim como os componentes e respetivos *top movers*[1].

A respeito da segurança, a MBolsa só permite o acesso após a validação de um PIN que deve ser exclusivamente do conhecimento do utilizador. O dispositivo é registado individualmente e associado ao seu perfil. A comunicação entre o cliente (terminal MBolsa) e o servidor (servidor Millennium bcp) é efetuada sobre canais seguros, utilizando mecanismos de encriptação por forma a assegurar a confidencialidade da comunicação.

A MBolsa, comparativamente ao SifoxDeal Mobile, apresenta algumas limitações, como por exemplo, a negociação exclusiva em mercados nacionais e a necessidade de abertura de conta junto do banco Millennium bcp. É uma aplicação orientada a utilizadores finais que gostam de negociar, tirando partido da sua conta bancária. O SifoxDeal Mobile é uma aplicação multi cliente desenhada e pensada tanto para a utilização profissional como para a utilização doméstica e que permite a consulta,

[1]Instrumentos financeiros que mais influenciam o mercado bolsista.

análise e negociação nos mercados financeiros nacionais e internacionais [4].

Capítulo 3

Aplicação

3.1 Padrões de Desenho de Software

Os Padrões de Desenho descrevem soluções para problemas comuns num determinado contexto. O contexto é a situação recorrente em que o padrão é aplicado. O problema é o objetivo a alcançar neste contexto, bem como quaisquer restrições associadas ao mesmo. A solução é o que se procura, um desenho flexível e modular de acordo com o contexto, que atinja os objetivos e resolva as restrições [7].

Esta ideia foi introduzida, pelo arquiteto Christopher Alexander, no ramo da Arquitetura. Alexander descreveu o uso de padrões em edifícios e áreas urbanas no lançamento do livro *"A Pattern Language: Towns, Buildings, Construction"* em 1977 [2]. Mais tarde, essa ideia foi expandida pelas mais diversas áreas, como a da Ciência de computadores.

Os Padrões de Desenho são ferramentas de abstração muito úteis no que respeita ao desenvolvimento de *Software*. *"Design Patterns: Elements of Reusable Object-Oriented Software"* é um livro que foi escrito por Erich Gama, John Vlissides, Ralph Jonhson e Richard Helm, mais conhecidos como *"The Gang of Four"*- GoF [22]. Neste livro os GoF descreveram vinte e três padrões de desenho de *software*, cada um constituído por um nome, um problema, uma solução e consequências. Os 23 padrões são normalmente agrupados em três categorias:

- **Padrões de Criação** – *Abstract Factory*, *Builder, Factory Method*, *Prototype, Singleton.*

- **Padrões de Estrutura** – *Adapter, Bridge*, *Composite*, *Decorator*, *Facade*, *Flyweight, Proxy.*

- **Padrões de Comportamento** – *Chain of Responsability*, *Command*, *Interpreter*, *Iterator*, *Mediator*, *Memento*, *Observer*, *State*, *Strategy*, *Template Method*, *Visitor.*

Na implementação do SifoxDeal Mobile foram utilizados três padrões de desenho, o padrão *Observer*, o padrão *Delegation* e o padrão *Model-View-Controller* (MVC). O padrão *Observer* foi descrito pelos GoF e classificado como padrão de comportamento [22].

3.1.1 O Padrão Observer

O padrão *Observer* define uma dependência de um-para-muitos entre objetos. Quando o estado de um objeto muda todos os seus dependentes são notificados e atualizados automaticamente.

Este padrão é muito útil, principalmente quando se utiliza o padrão de arquitetura MVC. Da camada *Model* fazem parte os objetos sujeitos, ou seja, aqueles que são observados. Da camada *View* fazem parte os *observers* ou *listeners*, ou seja, as interfaces de utilizador. Quando parte ou totalidade das interfaces fornecem dados da camada *Model* que pode mudar de estado, a função do sujeito, que mantém e gere uma lista de *observers*, é notificar todos os *observers* desta mudança, de modo a que os mesmos possam atualizar e mostrar a nova informação ao utilizador [13]. A figura 3.1 apresenta a arquitetura do padrão *observer.*

São quatros os componentes integrantes do padrão Observer:

- ***Subject*** – Interface que permite gerir os *observers*;

- ***ConcreteSubject*** – Notifica os *observers* da mudança de estado;

- ***Observer*** – Interface que permite a propagação da notificação aos *observers*;

- ***ConcreteObserver*** – Mantém uma referência para o objeto *ConcreteSubject* e implementa a interface *Observer.*

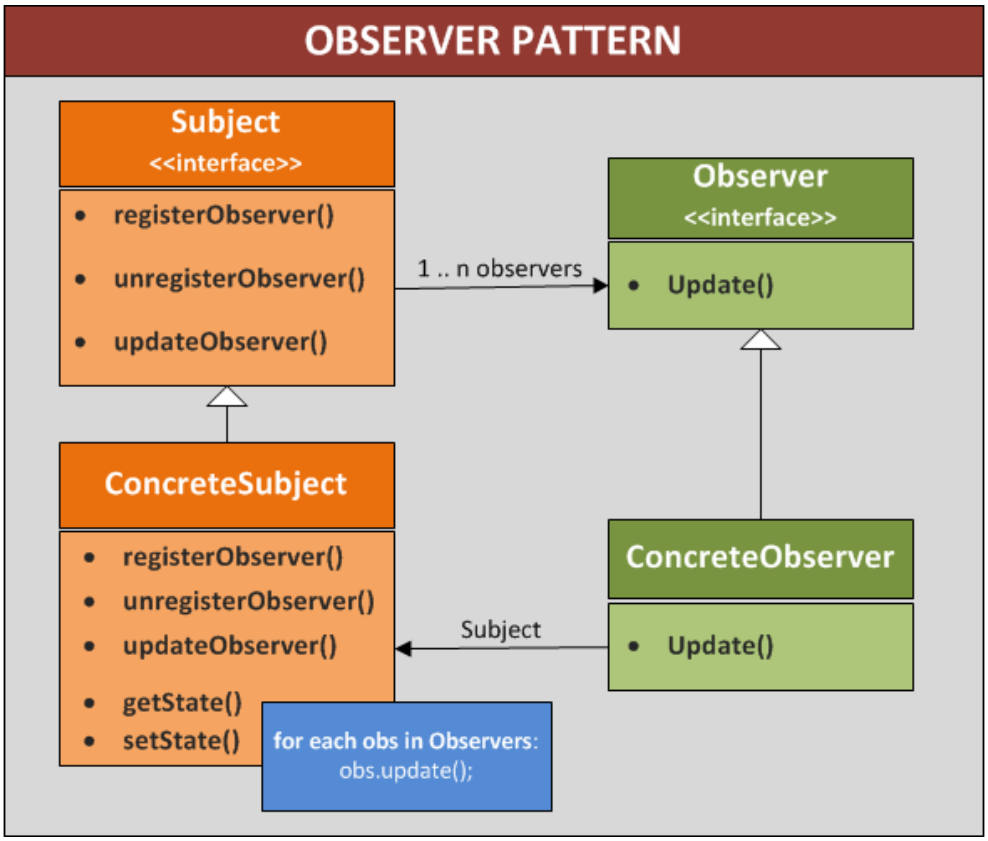

Figura 3.1: Arquitetura do Padrão Observer

3.1.2 O Padrão Delegation

O padrão *delegation* é um padrão de desenho utilizado em programação orientada a objetos, onde um objeto, chamado de *delegator*, delega, a outro objeto, chamado de *delegate object*, a responsabilidade de implementar e responder a um determinado método ou conjunto de métodos. É um mecanismo de abstração que, de certa forma, centraliza o comportamento do objeto.

Este padrão é uma das formas mais flexíveis de estabelecer uma relação entre duas classes. Muitos outros padrões de desenho utilizam o padrão *Delegation*. São eles, o padrão *the State*, o *Strategy* e o *Visitor*. A figura 3.2 apresenta a arquitetura do padrão *delegation*.

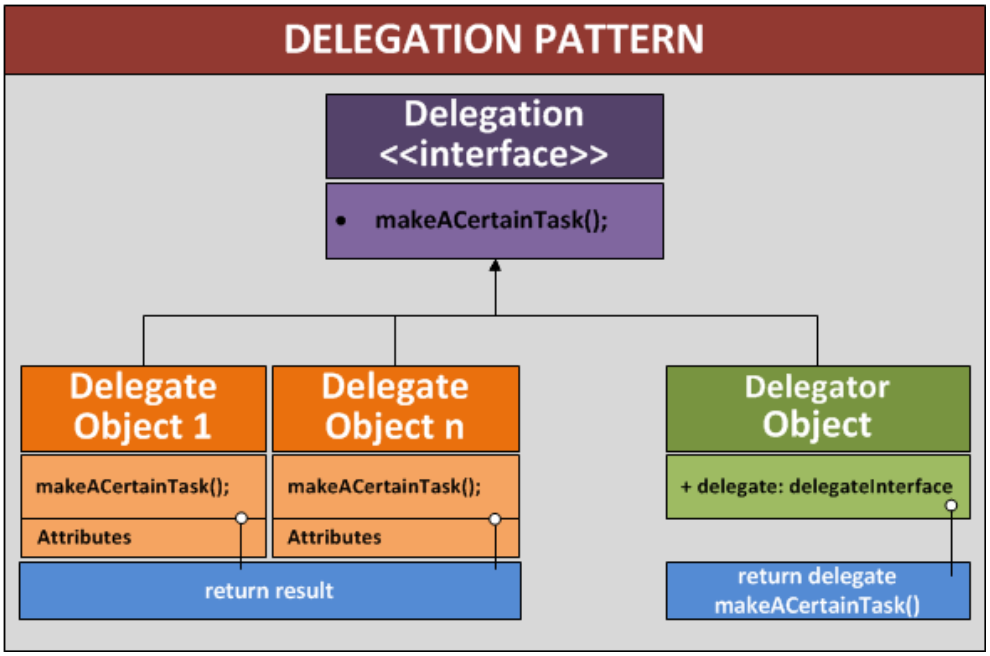

Figura 3.2: Arquitetura do Padrão Delegation.

3.1.3 Model-View-Controller

MVC é um padrão de alto nível de desenvolvimento de *software* que trata da arquitetura global da aplicação. Separa a parte lógica da aplicação da interface do utilizador, classificando os objetos de acordo com os métodos e o contexto.

Os objetos podem ser classificados em três tipos distintos: objetos *model*, *view* e *controller*. O padrão MVC define a função e a linha de comunicação de cada um desses tipos de objetos. Uma aplicação que implemente este padrão de arquitetura estará então dividida em três camadas: a camada *Model*, a camada *View* e a camada *Controller*. A arquitetura do padrão MVC é ilustrada pela figura 3.3.

A camada *Model* é uma representação detalhada e estruturada dos dados da aplicação. Define a lógica que manipula os dados. Uma boa implementação do padrão agrupa toda a informação em objetos *model*, principalmente aquela que faz parte do estado persistente da aplicação; a informação que é carregada para a aplicação e que tende a ser reutilizada. Notifica os *observers* (*views*) associados da mudança do seu estado, permitindo aos mesmos a atualização da sua interface. De um modo geral, a camada *Model* não se preocupa com questões de interface e apresentação.

A camada *View*, por sua vez, apresenta a camada *Model* da aplicação num formato adequado e estruturado ao utilizador, permitindo a edição a partir do model. O

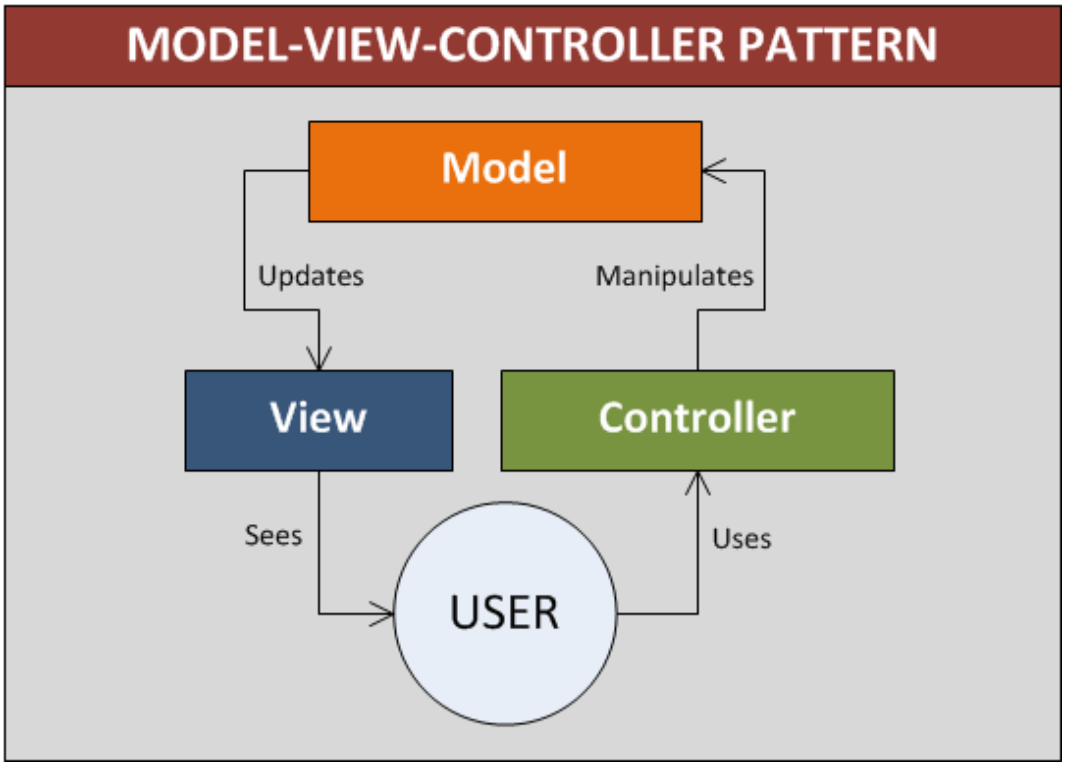

Figura 3.3: Arquitetura do Padrão Model-View-Controller.

armazenamento de informação não deve fazer parte da responsabilidade desta camada. Claro que existem exceções, como por exemplo, a camada *View*, por questões de *performance*, pode armazenar informação em *cache*. A *View* pode ainda ser responsável por mostrar uma parte de uma camada *Model*, toda uma camada *Model* ou até mesmo partes de camadas *Model* diferentes. Os objetos *view* tendem a ser flexíveis e reutilizáveis.

Uma *view* deve garantir a apresentação correta dos dados, por isso, é necessário que seja notificada das mudanças de estado que ocorrem na camada *Model*. Portanto, os objetos *model* não devem permanecer associados a objetos *view* específicos, necessitando de um modo genérico que lhes permita notificar as *views*.

A camada *Controller* é uma camada intermédia entre a camada *View* e a *Model* da aplicação. É da sua responsabilidade garantir o acesso dos objetos *view* à camada *Model* e agir como canal sobre o qual os mesmos podem realizar alterações. Os objetos *controller* podem também executar tarefas ou processos para uma aplicação e gerir o ciclo de vida de outros objetos [20].

O padrão MVC é um padrão composto por compreender diversos padrões elementares

como é o caso do padrão *observer* e o padrão *delegation* já descritos acima.

Esse padrão de arquitetura foi utilizado na implementação do SifoxDeal Mobile. Na subsecção seguinte é analisada e descrita toda a arquitetura da aplicação, assim como a caraterização dos objetos e do seu contexto de acordo com esse padrão de arquitetura.

3.2 Arquitetura

Nesta secção será descrita e esquematizada a arquiterura da aplicação, quer no seu contexto global, quer no seu contexto mais específico, no que respeita à estrutura e interação entre cada *view controller* da aplicação e a sua respetiva camada de tratamentro e gestão da informação (camada *model*). É de realçar que a arquitetura apresentada reflete o desenho, a estrutura e toda a lógica de implementação com a interface de utilizador, uma vez que a restante insfrastrutura (*Middle Office* e *Back Office*) já se encontrava desenvolvida pela Finantech.

A arquitetura geral do SifoxDeal Mobile é apresentada pela figura 3.4.

3.2.1 ServerDeal

O *ServerDeal* é um servidor que não se encontra esquematizado na arquiterura global da aplicação, uma vez que não esteve diretamente envolvido no processo de desenvolvimento deste projeto. Contudo, pela razão de se tratar de um componente essencial e indispensável ao bom funcionamento da aplicação, é importante conhecermos um pouco do seu funcionamento. Portanto, o *ServerDeal* é um servidor que se liga a uma base de dados *Oracle* e que tem como principal objetivo fornecer informações dos mercados financeiros aos demais clientes *SifoxDeal* através do protocolo TCP/IP. Este servidor, normalmente, pode estar ligado a três tipos de entidades. À entidade *Middle Office*, onde se encontra toda a informação relativa aos clientes, à entidade sistemas externos como a *GL*, *Euronext* ou *Visual Trader*, a fim de receber informações dos mercados e, por fim, a uma entidade que pode ser um outro *ServerDeal*.

O servidor pode enviar e receber informações de duas formas diferentes. Em *feed*, caso em que os Sistemas Externos ou o *Middle Office* enviam continuamente informações de negociação. Ou a pedido, caso em que os Sistemas Externos ou o *Middle Office* processem um pedido vindo do servidor, enviando depois a resposta.

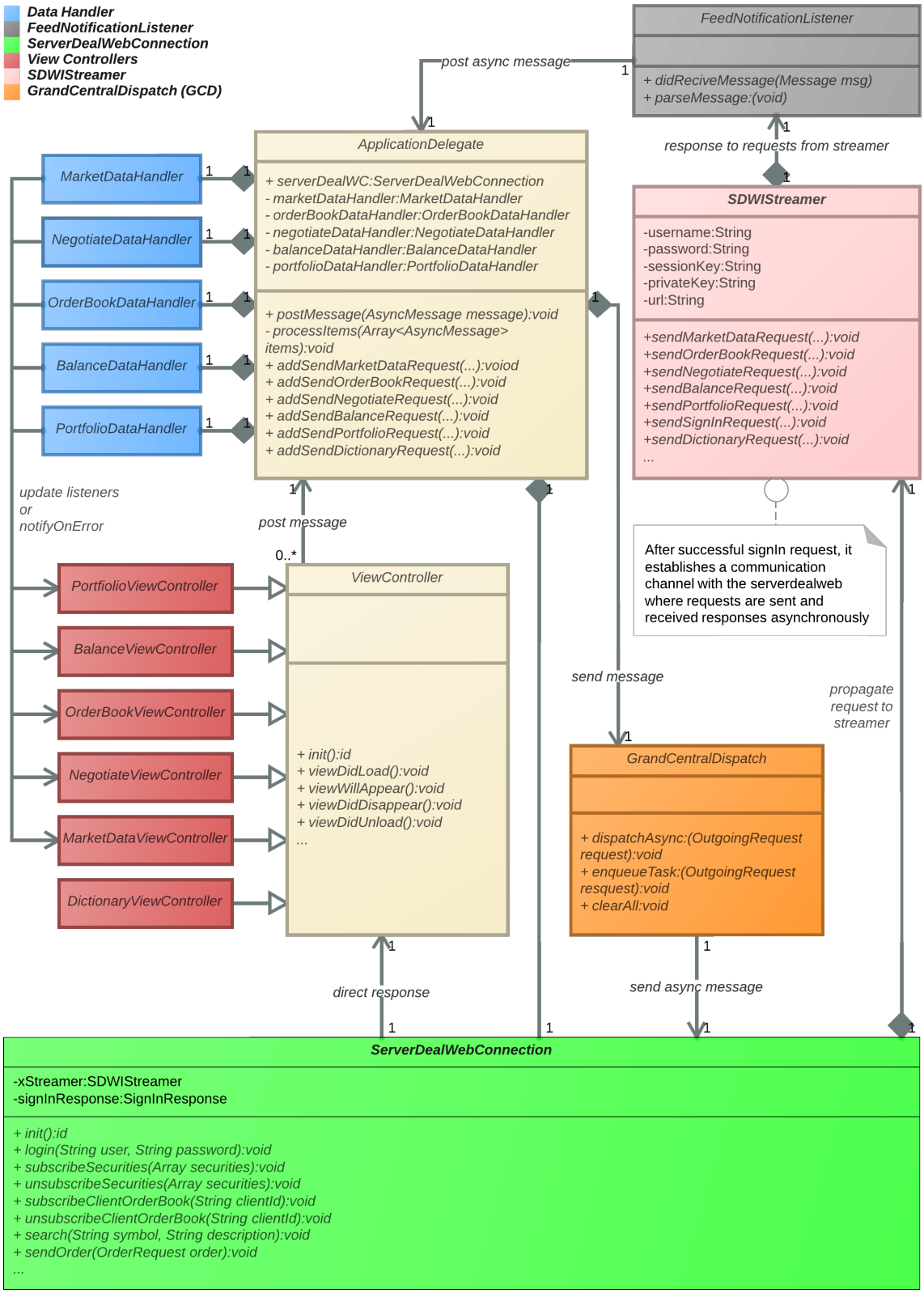

Figura 3.4: Arquitetura do SifoxDeal Mobile.

3.2.2 ServerDealWeb

O *ServerDealWeb* à semelhança do *ServerDeal* não se encontra esquematizado na arquitetura global da aplicação. Este servidor surge com a necessidade de fornecer toda e qualquer informação disponibilizada pelo *ServerDeal* a utilizadores na Internet. Portanto, o *ServerDeal* Web funciona como uma espécie de elo de ligação entre o *ServerDeal*, que funciona sob o protocolo *TCP/IP – FIX*, e o mundo Web, protocolo *HTTP*. Logo, o servidor que se encontra mais próximo da aplicação é o *ServerDealWeb*, mais próximo porque é o servidor que comunica diretamente com a aplicação através de um *streamer* designado por *SDWIStreamer*.

O *ServerDealWeb*, também através de *web services*, está ligado a um núcleo central, a fim de validar toda e qualquer informação de um determinado cliente, assim como as suas configurações e definições.

3.2.3 SDWIStreamer

O *SDWIStreamer* é uma biblioteca estática que contém um conjunto de métodos que permitem a receção em *real time* da informação, como é o caso dos dados de mercado. Este resultado é obtido pela ligação do *SDWIStreamer* ao *ServerDealWeb*. Tudo é gerido sob o padrão *publish - subscribe* e comunicações assíncronas. Basicamente, depois da subscrição de um certo tipo de informação, é retornada a devida resposta em *real time*, e sempre que haja nova informação do mesmo tipo, será enviada uma mensagem *feed* de actualização assíncrona sem que seja necessário pedir o mesmo tipo de informação.

Para criar uma instância da classe *SDWIStreamer* são necessários dois parâmetros. O primeiro é o *URL* do *ServerDealWeb*. O segundo parâmetro é a referência para um objeto que implementa a interface *ISDWIStreamer*, que no caso do *SifoxDealMobile* trata-se de um objeto da classe *FeedNotificationListener*.

3.2.4 FeedNotificationListener

O *FeedNotificationListener* é o objeto passado como referência ao *SDWIStreamer* e que implementa a *interface ISDWIStreamer*. É um objeto que permanece à escuta, a fim de receber as novas mensagens de *feed*. As mensagens, após um pedido, podem ser de vários tipos, enumerados em seguida:

- **MDFt** – Market Data Feed;
- **OBF** – OrderBook Feed;
- **PF** – Portfolio Feed;
- **BF** – Balance Feed;
- **NOF** – New Order Feed;
- **COF** – Cancel Order Feed;
- **ROF** – Replace Order Feed;
- **NF** – News Feed;
- **CIF** – ClientIds Feed;

O *FeedNotificationListener* implementa os seguintes métodos de acordo com a *interface* do *streamer*:

- **didReceiveMessage:(Message *)message** – Este método é o mais importante do *FeedNotificationListener* e é executado perante a chegada de uma mensagem *feed*. Neste método obtém-se o tipo da mensagem e organiza-se a informação numa estrutura de dados que depois é enviada à camada *model* da aplicação.
- **didFailWithError:(NSError *)error** – Método que é invocado, aquando da ocorrência de um erro, como por exemplo, a perda de ligação à rede.
- **didFinishLoading** – Método que indica o fim do *loading* de uma mensagem *feed*.
- **reconnect:(NSError *)error** – Método que, após o surgimento de um erro reportado pelo método *didfFailWithError:* referente à perda de ligação da aplicação, tenta estabelecer a ligação novamente várias vezes.

3.2.5 ServerDealWebConnection

ServerDealWebConnection é a classe que, instânciada, expõe aos objetos interessados todas as operações do *streamer SDWIStreamer*. Quer isto dizer, que esta classe mantém uma instância do *streamer*, que permite a receção e o respectivo encaminhamento de todos os pedidos assíncronos que chegam com destino ao *ServerDealWeb*.

É aqui que também são recebidas as respotas directas a alguns pedidos que não chegam por messagens *feed*, pelo simples fato de se tratarem de pedidos de consulta de informação estática e esporádica.

Resumindo, a classe *ServerDealWebConnection* funciona como uma espécie de *interface* de ligação entre a aplicação e o *ServerDealWeb* através do *streamer*, tornando-se responsável por todo e qualquer pedido ao servidor.

3.2.6 ApplicationDelegate

O objeto *applicationDelegate* é um objeto criado aquando da execução da aplicação, normalmente, pela funcão *main* da aplicação. É o objeto "coração"da aplicação, uma espécie de "localização central", que controla a aplicação e os seus eventos de acordo com o protocolo *UIApplicationDelegate*. Os métodos mais importantes que o protocolo declara e que são implementados pelo *applicationDelegate* são so seguintes:

- application:willFinishLaunchingWithOptions:
- applicationDidBecomeActive:
- applicationDidEnterBackground:
- applicationWillEnterForeground:
- applicationDidFinishLaunching:
- applicationWillTerminate:

Por se tratar de um objeto "central"da aplicação, é aqui mantida uma instância de cada um dos *data handlers* e do *ServerDealWebConnection*, podendo facilmente interagir com outros objetos *view* ou objetos *model* da aplicação.

3.2.7 GrandCentralDispatch (GCD)

A designação desta classe, *GrandCentralDispatch*, advém da tecnologia desenvolvida pela *Apple*, que tem exatamente este nome, com o principal objetivo de tornar o uso de *threads* e mecanismos de sincronização tradicionais mais transparentes para o programador, e ao mesmo tempo, possibilitar a execução de tarefas em paralelo mais simples e eficiente.

Portanto, esta classe faz uso desta tecnologia, que mantém e gere uma fila *FIFO* de tarefas que são inseridas num contexto *block* (conjunto de instruções a executar), e que podem ser executadas noutra ocasião. A *FIFO* é gerida automaticamente pela tecnologia, sendo completamente transparente ao programador.

Assim sendo, compreende-se que todo e qualquer pedido que é enviado ao *ServerDealWeb*, através do *SDWIStreamer*, é inserido num contexto *block* de instruções a executar, colocado na fila *FIFO*, e executado posteriormente de forma assíncrona.

A tecnologia GCD será descrita mais à frente detalhadamente no capítulo das Tecnologias e Ferramentas Utilizadas.

3.2.8 Data Handler dos Dados de Mercado e os seus listeners

A arquitetura e a lógica de interação entre a camada *model* dos dados de mercado e os seus *view controllers* (*listeners*) podem ser esquematizadas e representadas por quatro componentes ou áreas essenciais. O *Data Handler*, que manipula toda informação referente a cada título de mercado. O *Data Formatter*, que tem como função formatar qualquer tipo de valor numérico associado a cada título. O *Data Helper*, que proporciona ao *Data Handler* funções auxiliares, mas essenciais. E, por fim, o *User Interface* ou conjunto de *views* onde é mostrada ao utilizador toda a informação oriunda do *Data Handler*, de forma organizada e estruturada. Esta arquitetura é apresentada pela figura 3.5.

3.2.8.1 Data Handler

- **Securities Handler** – O *Securities Handler* mantém em memória uma lista de títulos de mercado. Cada um desses títulos é representado por um objeto *securityHandler* (instância da classe *Security Handler*).

  Uma estrutura de dados do tipo *hash map*[1] foi utilizada para armazenar a informação de cada título. A escolha da utilização de uma *hash* para armazenar a informação deve-se principalmente à simplicidade de implementação, por ser indicada para armazenar grande volume de dados, como é o caso dos dados de mercado, e permitir pesquisas rápidas. Cada item de uma *hash* é composto pelo

[1]Estrutura de dados que utilizada uma função para associar chaves a valores, permitindo a pesquisa.

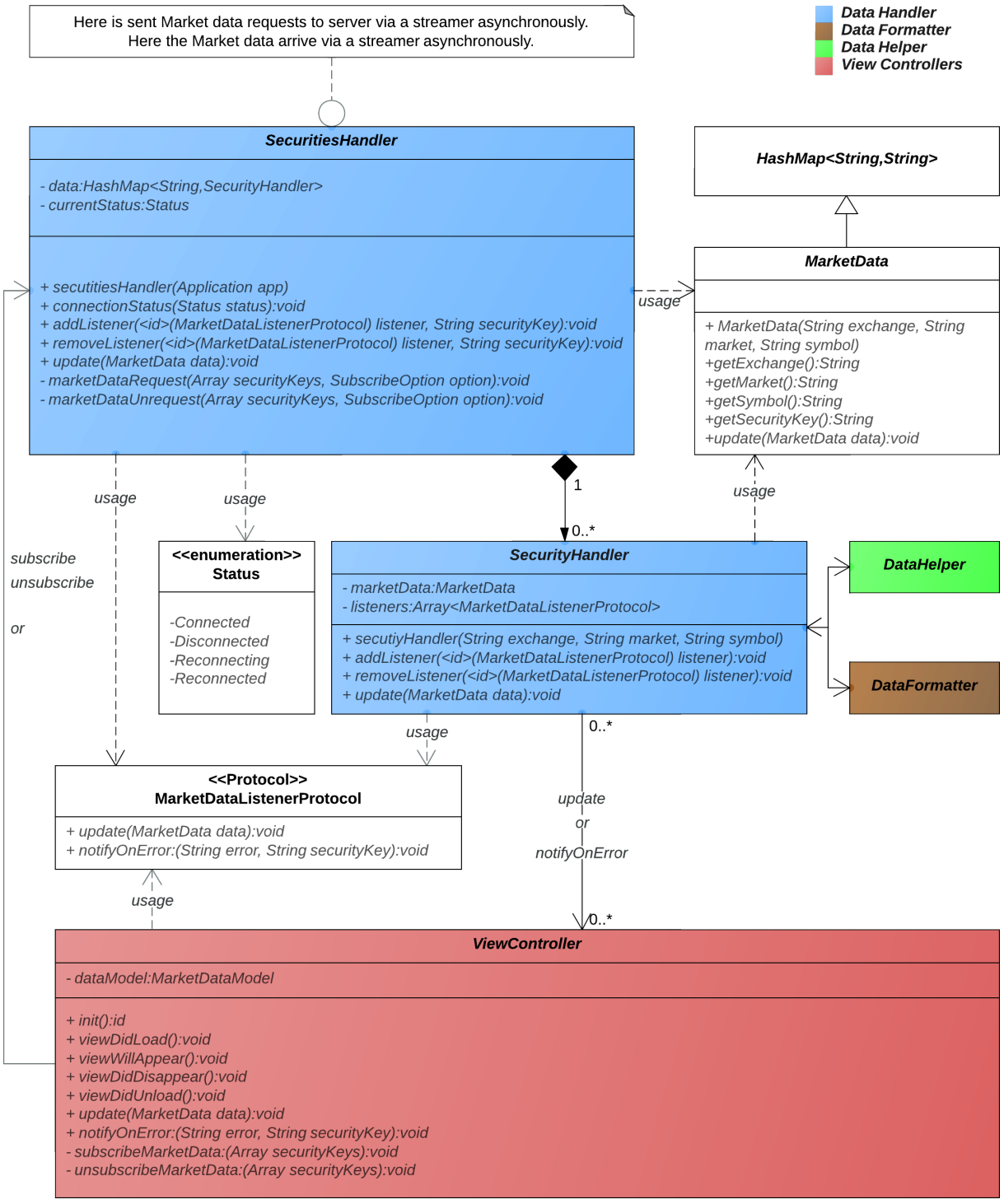

Figura 3.5: Arquitetura do data handler dos Dados de Mercado e a interação com os seus listeners.

par *chave-valor*. Neste caso, a chave é uma string[1], denominada *SecurityKey*, e o seu valor, um objeto *securityHandler*.

SecurityKey é uma string única que representa e descreve, de certa forma, um título pertencente a uma determinada praça e mercado. Por exemplo, o título EDP da praça Lisboa, pertencente ao índice *PSI20*, é dado como *"XLIS+CS+EDP"*. Posto isto, é de notar que o *Securities Handler* mantém e gere uma *hash* do tipo:

```
data = {SecurityKey1:SecurityHandler,...,SecurityKeyN:SecurityHandlerN}
```

- **Security Handler** - O *Security Handler* é um objeto que representa e armazena toda a informação referente a um título. Este objeto mantém, por isso, uma variável de instância do tipo *MarketData*, que mais não é do que um objeto que estende uma estrutura de dados do tipo *hash map*, onde é guardada toda a informação do título de mercado em questão. O objeto, apesar de ser um mero "transportador" de informação, expõe alguns métodos que permitem a atualização e o acesso da informação. Cada entrada da *hash map* do objeto é formada por uma chave, que se trata de um valor inteiro representado em forma de *string*, e um valor, que pode ser quantitativo ou semântico, é também representado em forma de *string*:

  ```
  Chave:‘‘103’’ -> Valor:‘‘EDP’’

  Chave:‘‘10’’ -> Valor:‘‘1,967’’
  ```

  A Chave "103", por exemplo, corresponde ao nome do título e a chave "104" ao preço atual do mesmo.

  O *Security Handler* mantém e gere também uma lista de *observers* ou *listeners*. Aqui está subjacente a utilização do padrão de desenho *Observer* e *Delegation* já descritos com algum detalhe na subsecção 3.1.1 e 3.1.2 respetivamente.

- **Incoming Messages**

  - ***Update*** – Mensagem de atualização de dados de mercado referente a um determinado título. Se for a primeira mensagem após o pedido, a mesma acarreta toda a informação disponível sobre o título, caso contrário, só os campos que sofreram alterações serão enviados e posteriormente atualizados.

[1]Em programação, uma string é um tipo de dados que representa uma sequência de caracteres.

- ***Add Listener*** – É uma mensagem enviada por um objeto que pertence à camada *View* da aplicação, subjacente à aplicação do padrão de arquitetura MVC, a um objeto *model* (*securityHandler*). O objeto *view* passa a partir deste momento a ser notificado por qualquer mudança do estado do objeto *model* que o subscreveu.
- ***Remove Listener*** – A mensagem *Remove Listener* é a mensagem que permite a um objeto *View* libertar-se das notificações de mudança do estado de um determinado objeto *model* ao qual foi subscrito. É a lógica inversa da mensagem *Add Listener*.
- ***Connection Status*** – É uma mensagem que se forma aquando da mudança do estado da ligação da aplicação. Existem diversos estados possíveis definidos na implementação do *SifoxDeal Mobile*. O estado *Disconnected*, *Connected*, *Reconnecting* e *Reconnected*.

- **Outgoing Messages**

  - ***Market Data Request*** – Mensagem de pedido de dados de mercado ao servidor. É enviado ao servidor uma lista de *securityKeys* e o tipo de subscrição, que neste caso é de *subscribe*.
  - ***Market Data Unrequest*** – Mensagem de pedido de dados de mercado ao servidor. É enviado ao servidor uma lista de *securityKeys*, mas neste caso o tipo de subscrição é *unsubscribe*.

3.2.8.2 Data Formatter

O *Data Formatter* tem como função formatar qualquer tipo de dado numérico. Cada *Security Handler*, que inicialmente recebe informação não formatada de dados de mercado referente a um título, envia mensagens ao *Formatter* de modo a obter uma formatação correta dos dados. Essencialmente são formatadas quantidades e preços de acordo com a localização geográfica do *device*[1]. O *Data Formatter* não é utilizado apenas pela arquitetura dos dados de mercado, por isso é composto por métodos estáticos que possibilitam a formatação adequada dos dados.

[1]Dispositivo móvel.

3.2.8.3 Data Helper

É da responsabilidade do *Data Helper* fornecer métodos auxiliares a objetos do tipo *Security Handler* que permitem, por exemplo, a edição de *strings* e cálculos auxiliares.

3.2.9 Data Handler do Livro de Ordens e os seus listeners

A lógica arquitetural do *data handler* do Livro de Ordens com os seus *listeners* é em muito semelhante à dos dados de mercado. É também composta por quatro componentes essenciais. O *Data Handler*, que manipula toda a informação relativa às ordens agrupadas por cliente. O *Data Formatter*, que tal como na arquitetura de dados de mercado, tem como função formatar valores numéricos, mas neste caso referentes a uma ordem. O *Data Helper*, que proporciona ao *Data Handler* funções auxiliares, mas essenciais. E, por fim, o *User Interface*. Uma visão global desta arquitetura pode ser obtida pela figura 3.6.

3.2.9.1 Data Handler

- **OrderBookDataHandler** – O objeto *orderBookDataHandler* mantém em memória uma lista de clientes. A cada cliente está associado um identificador denominado *clientId*[1]. E para cada *clientId* existe um objeto *clientOrders*[2].

  Pelas razões já mencionadas, aquando da descrição da arquitetura dos dados de mercado, a estrutura de dados utilizada foi uma *hash map*. Neste caso, cada item da *hash* é formado pelo par chave-valor *clientId* e objeto *clientOrders* respetivamente. Então, o *orderBookDataHandler* mantém e gere uma *hash* do tipo:

  ```
  data = {ClientId1:ClientOrders1,...,ClientIdN:ClientOrdersN}}
  ```

- **ClientOrders** - O objeto *clientOrders* representa e armazena toda a informação das ordens de um cliente. Mantém, por isso, uma *hash* do tipo:

  ```
  data = {OrderBookKey1:ClientOrder1,...,OrderBookKeyN:ClientOrderN}}
  ```

[1]É um identificador do cliente.
[2]É uma instância (objeto) da classe ClientOrders.

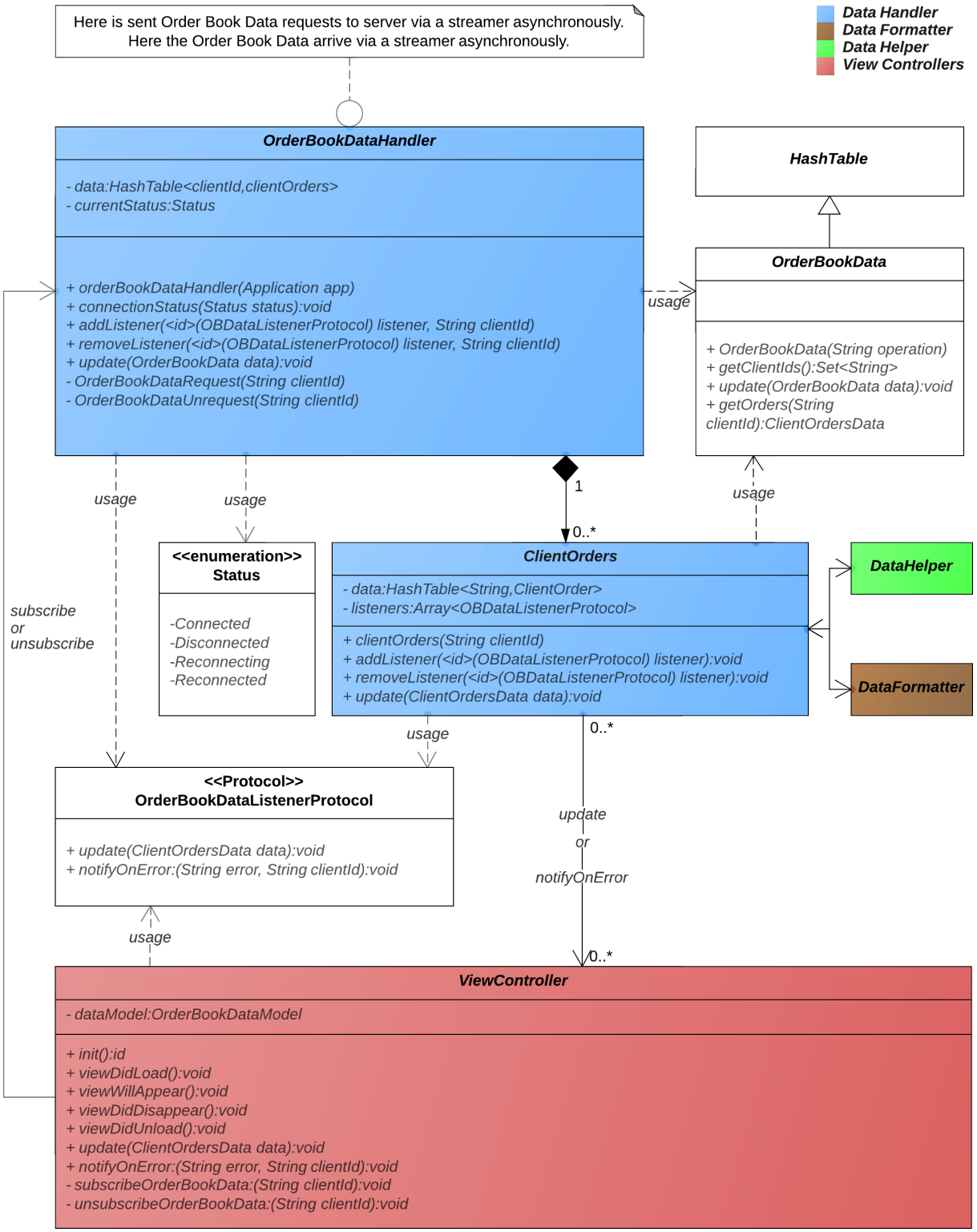

Figura 3.6: Arquitetura do data handler do Livro de Ordens e os seus listeners.

OrderBookKey é uma string única formada pela concatenação da data e da hora do envio da ordem, mais um número inteiro sequencial. É utilizada como chave de cada ordem. Por sua vez, o objeto *clientOrder*, que estende uma estrutura de dados do tipo *hash map*, armazena toda a informação de uma ordem.

O objeto *clientOrders* também mantém e gere uma lista de *listeners*.

- **Incoming Messages**
  - **Update** - A mensagem *update* surge da necessidade de atualizar a informação de uma ordem ou conjunto de ordens respetivas a um cliente. As razões desta necessidade são diversas, seja porque a ordem foi executada, ou porque foi suspensa ou anulada, bloqueada, modificada, ou seja, por qualquer que seja a mudança do seu estado inicial. A primeira mensagem de *update* transporta toda a informação da ordem.
  - **Add Listener** - É uma mensagem enviada por um objeto *view* da aplicação com o objetivo de se tornar *"listener"* de um objeto *clientOrders* associado a um *clientId*.
  - **Remove Listener** - O objeto *view* que envia a mensagem deixa de ser notificado das mudanças de estado do objeto *clientOrders* a que foi subscrito.
  - **Connection Status** - Mensagem gerada pela mudança do estado da ligação da aplicação.
- **Outgoing Messages**
  - **Order Book Data Request** - Mensagem de requisição de toda a informação referente às ordens de um cliente. A mensagem é composta pelo *clientId* e pelo tipo de subscrição.
  - **Order Book Data Unrequest** - Mensagem de pedido de *unsubscribe* da informação das ordens de um cliente.

3.2.10 Pesquisa no Dicionário de títulos

Nesta subsecção é apresentada a lógica do funcionamento de uma pesquisa de títulos no dicionário por parte de um objecto *view* da aplicação. Esta representação, a nível da arquiterura, é bem mais simples que as arquiteturas já descritas anteriormente. É composta por três componentes essenciais. Dois dos componentes são objetos que pertencem à camada *View* da aplicação. São eles o *dictionaryViewController*,

objeto principal da arquitetura, e um outro objeto *viewController* que, por ação do utilizador, inicia e usufrui da pesquisa de títulos. O terceiro componente é a interface (*ServerDealWebConnection*) de ligação ao servidor *ServerDealWeb*. A figura 3.7 ilustra esta lógica do funcionanmento do dicionário de títulos.

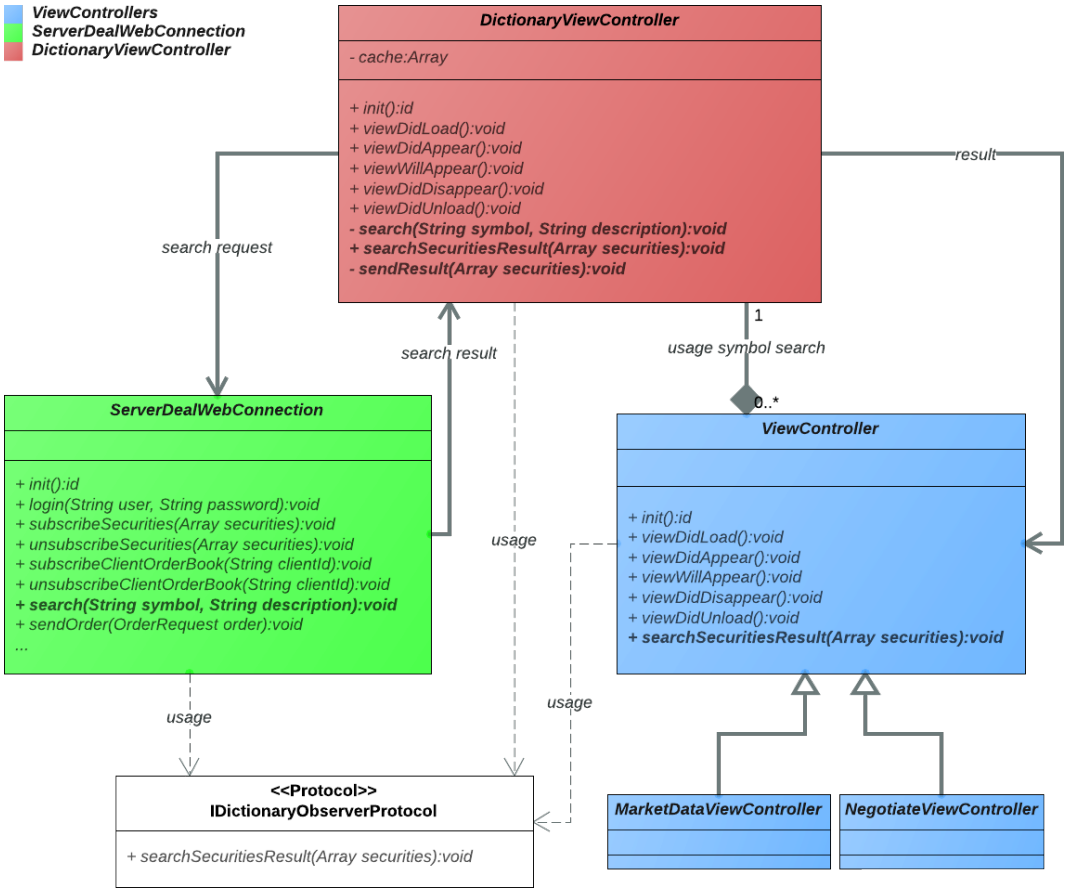

Figura 3.7: Arquitetura da pesquisa no Dicionário de títulos.

Uma nova instância do *dictionaryViewController* é criada, através de um outro objeto *viewController*, pela interação do utilizador, sempre que o mesmo pretenda usufruir da pesquisa em *live search* de títulos de qualquer mercado e praça.

No pedido de pesquisa é enviado uma *string* que, de certa forma, representa o título a pesquisar, que pode ser o *symbol* do título, ou então, uma *string* mais extensa que descreva o mesmo. Por exemplo, o título com o *symbol* AAPL, da praça XLIS e mercado CS, referente à empresa *Apple*, pode ser pesquisado pelo seu *symbol* AAPL, ou então, por uma descrição do género *"Apple In"*.

No pedido é também enviada uma referência ao objeto *dictionaryViewController* que implementa o protocolo *IDictionaryObserverProtocol*. Um protocolo, de modo

geral, define um conjunto de métodos que podem ser implementados por qualquer classe. Neste caso, o protocolo *IDictionaryObserverProtocol* declara um método - *searchSecuritiesResult:* com um conjunto de títulos (*securities*) como argumento:

```
@protocol IDictionaryObserverProtocol <NSObject>
@required
-(void)searchSecuritiesResult:(NSArray *)securities;
@end
```

Após o pedido ter sido processado pelo servidor, é enviada uma resposta direta (*Direct Response*) ao objeto *dictionaryViewController* com o resultado de acordo com o protocolo *IDictionaryObserverProtocol*. No caso do servidor não encontrar títulos associados ao *symbol* ou à descrição, será passado no método um *array* de tamanho zero.

Depois do utilizador visualizar os resultados da pesquisa, ele possui a opção de selecionar parte ou totalidade dos títulos encontrados por forma a enviá-los ao objeto *view* que solicitou a pequisa. Este objeto também implementa o método do protocolo *IDictionaryObserverProtocol* a fim de receber o resultado da pesquisa que, posteriormente, serão utilizados com alguma finalidade.

3.2.11 Negociar

O processo de negociação disponibilizado pela aplicação envolve três entidades essencias. O objeto *view*, que disponibiliza ao utilizador uma interface de negociação. Um objeto *model*, que armazena e gere toda a informação da ordem a enviar. E o objeto *serverDealWebConnection*, que trata do envio dos pedidos de negociação ao servidor de forma assíncrona com auxílio ao *SDWIStreamer*. Este processo de negociação encontra-se esquematizado pela figura 3.8.

NegotiateViewController - O *negotiateViewController* é um objeto da camada *View* da aplicação que permite ao utilizador o envio de ordens. O utilizador fornece um conjunto de informações essenciais, algumas obrigatórias, no ato do envio da ordem. Informações como, o título, a quantidade, o preço, a data de expiração e o *clientId* do cliente que envia a ordem.

Todas essas informações são armazenadas como propriedades num objeto designado

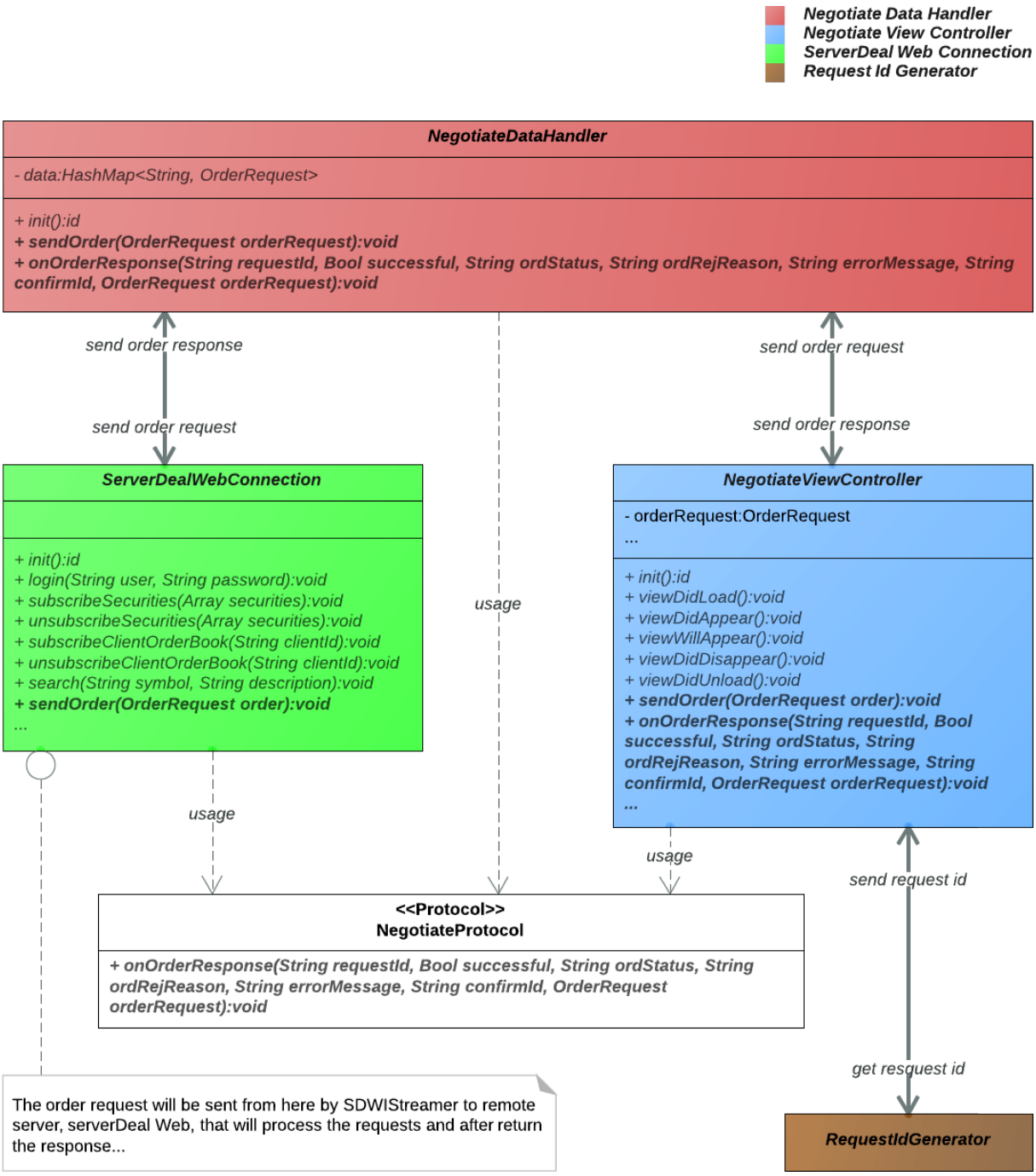

Figura 3.8: Arquitetura do processo de Negociação.

por *orderRequest*[1]. As propriedades fornecem um modo simples de declarar e implementar os métodos *get* e *set* de um objeto. Ao declarar, por exemplo, a string *clientId*

[1]Instância da classe *OrderRequest* que mantém a informação de uma ordem com destino ao mercado financeiro.

como propriedade, estes métodos serão automaticamente criados pelo sistema:

```
@property (nonatomic, strong) NSString *clientId;
```

Após o objeto *orderRequest* ser criado é-lhe associado, com auxílio ao objeto *requestIdGenerator*, um *id* (*requestId*) para que o pedido possa ser, de alguma forma, identificado. Posto isto, é então efetuado o pedido através da mensagem *sendOrder: orderRequest* ao objeto *model negotiateDataHandler*.

NegotiateDataHandler - O objeto *negotiateDataHandler* mantém em memória e gere toda a informação das ordens enviadas. Mantém uma *hash map* do tipo:

```
data = {requestId1:orderRequest1,...,requestIdN:orderRequestN}
```

Quando o *negotiateDataHandler* recebe a mensagem de envio de uma ordem, obtém a informação do *requestId* contida no objeto *orderRequest*, para que possa amazenar e identificar a ordem (*orderRequest*) temporariamente. Depois, a mensagem é então propagada ao objeto *serverDealWebConnection*.

ServerDealWebConnection - Neste contexto, a função do *serverDealWebConnection* passa apenas por encaminhar o pedido ao *ServerDealWeb*. O *ServerDealWeb*, por sua vez, tem a responsabilidade de processar o pedido e enviar a ordem para o mercado. Se, por alguma razão, o envio da ordem falhar, é enviada uma mensagem pelo objeto *serverDealWebConnection* ao *negotiateDataHandler* através de uma resposta direta, subjacente à implementação do protocolo *NegotiateProtocol*. Este protocolo declara um método do tipo:

```
-(void)notifyUpdate:(NSString *)requestId successfull:(NSString *)successFull
 ordStatus:(NSString *)ordStatus ordRejReason:(NSString *)ordRejReason
 errorMessage:(NSString *)errorMessage confirmId:(NSString *)confirmId
orderRequest:(OrderRequest *)orderRequest;
```

Este método, implementado pelo *negotiateDataHandler*, retorna a ordem rejeitada (*orderRequest*), o seu *requestId* e toda a informação relativa ao estado da ordem que depois é propagada ao objeto *negotiateViewController*, de modo a que a informação de falha e a devida justificação sejam apresentadas ao utilizador.

3.2.12 Data Handler do Financeiro e os seus listeners

A arquitetura e o fluxo de mensagens de comunicação entre o *data handler* do financeiro e os seus *listeners* são semelhantes às dos dados de mercado e do livro de ordens. São quatro os componentes principais desta arquitetura. O *Data Handler*, que nesta arquitetura possui a responsabilidade de gerir e manter toda a informação referente às contas bancárias dos clientes. O *Data Formatter*, que formata todos os dados numéricos necessários das contas de cada cliente. O *Data Helper*, que "ajuda" o *Data Handler* com métodos auxiliares. E o quarto componente, o *User Interface* (*view controller*). A arquitetura e o fluxo de mensagens podem ser observados através da figura 3.9.

3.2.12.1 Data Handler

- **BalanceDataHandler** – O objeto *balanceDataHandler* armazena em memória uma lista de contas bancárias. A cada conta está associado um *id* (*accountId*[1]). Esse *accountId* forma a chave de cada item da *hash map*, que, por sua vez, está associado a um objeto do tipo *balanceAccount*. A *hash map* utilizada, e que serve de suporte à memória, é do tipo:

  ```
  data = {accountId1:balanceAccount1,...,accountIdN:balanceAccountN}
  ```

- **BalanceAccount** – O objeto *balanceAccount* mantém, como variável de instância, um objeto do tipo *BalanceAccountData*. Este objeto estende uma estrutura de dados do tipo *hash map* e mantém toda a informação que compõe uma conta bancária de um cliente. O objeto estende, por isso, uma *hash* do tipo:

  ```
  data = {tag1:value1,...,tagN:valueN}
  ```

  Ambos valores, *tag* e *value*, são objetos *string* e representam a informação de um campo da conta bancária de um cliente. Exemplos de pares *tag-value* são mostrados a seguir:

  ```
  tag1 = ‘‘280’’ -> value1 = ‘‘20110219-EUR-CA’’
  tag2 = ‘‘281’’ -> value2 = ‘‘312,400’’
  ```

[1]Identificador da conta de um determinado cliente.

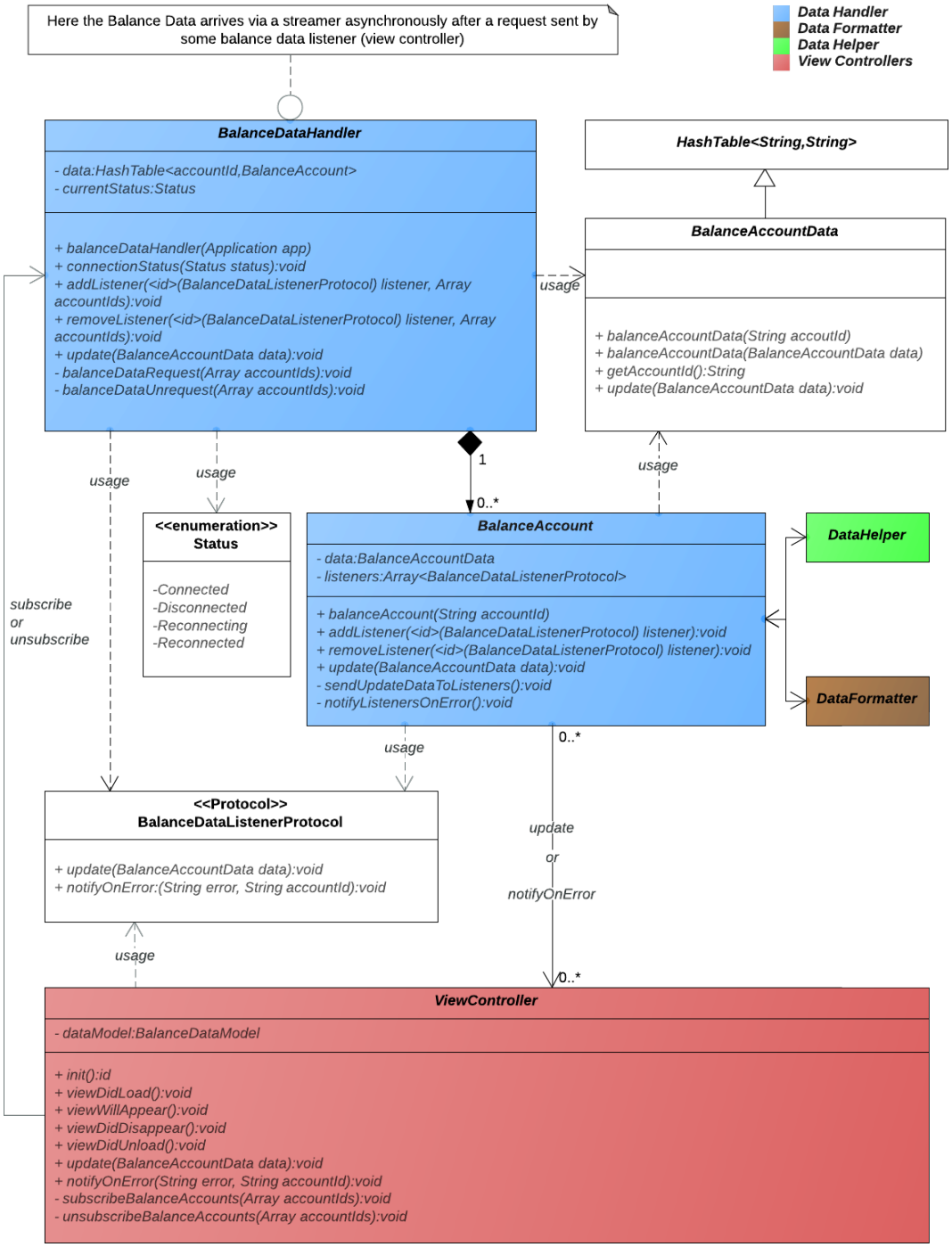

Figura 3.9: Lógica arquitetural do data handler do Financeiro e a comunicação com os seus listeners.

A *tag1*, com o valor "280", representa o nome da conta, e a *tag2*, com o valor "281", representa o saldo líquido da conta bancária.

O objeto *balanceAccount* também mantém e gere uma lista de *listeners*.

- **Incoming Messages**
  - **Update** – É uma mensagem de atualização da informação referente a uma conta bancária de um cliente. A primeira mensagem de *update* recebida pelo *balanceDataHandler* possui toda a informação possível e descritiva acerca de uma conta bancária. Se não for este o caso, só os campos que sofreram alterações serão enviados e posteriormente atualizados.
  - **Add Listener** – É uma mensagem enviada por um objeto que pertence à camada *View* da aplicação subjacente à aplicação do padrão de arquitetura *MVC*, a um objeto *model* (*balanceDataHandler*). O objeto *view* passa, a partir deste momento, a ser notificado por qualquer mudança do estado do objeto *model* que subscreveu.
  - **Remove Listener** – A mensagem *Remove Listener* é a mensagem que permite a um objeto *view* libertar-se das notificações de mudança do estado de um determinado objeto *model* que subscreveu. É a lógica inversa da mensagem *Add Listener*.
  - **Connection Status** – É uma mensagem que se forma aquando da mudança do estado da ligação da aplicação.
- **Outgoing Messages**
  - **BalanceDataRequest** – Mensagem de pedido de informação de uma certa conta bancária. É enviado ao servidor uma string *accountId* e o tipo de subscrição, que neste caso é de *subscribe*. O objeto *balanceDataHandler* passa a receber mensagens de *update* referentes ao *accountId* enviado no pedido.
  - **BalanceDataUnrequest** – Mensagem de *unsubscribe* de um *accountId*. O objeto *balanceDataHandler* deixa de receber mensagens de *updade* relativas ao *accountId* enviado no pedido.

3.2.13 Data Handler da Carteira e os seus listeners

A arquitetura e lógica organizacional do *data handler* da carteira e dos seus *listeners* são semelhantes às dos dados de mercado, livro de ordens e financeiro. Constituída por quatro componentes, são eles: *Data Handler*, *Data Formatter*, *Data Helper* e o *User Interface*. O *Data Handler*, que mantém em memória e gere um conjunto de títulos de mercado da carteira de um cliente. O *Data Fomatter*, que fornece métodos

auxiliares de formatação de valores numéricos respetivos a cada título da carteira de um cliente. O *Data Helper*, que fornece métodos auxiliares ao *Data Handler*. E o *User Interface*, que mostra de forma organizada os títulos em carteira de um cliente ao utilizador. Na figura 3.10 é mostrada a organização e as trocas de mensagens entre o *data handler* e os seus *listeners*.

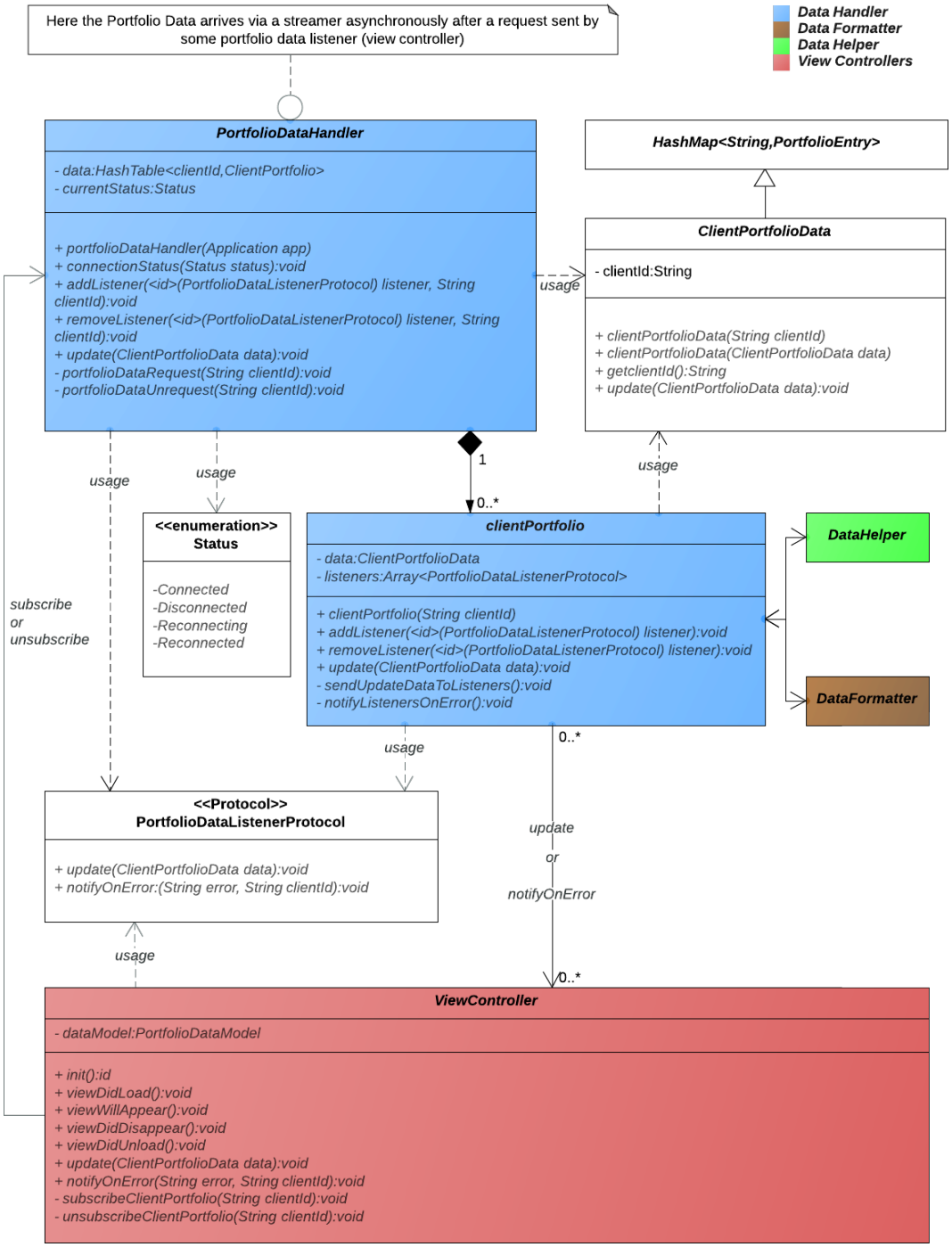

Figura 3.10: Lógica organizacional do *data handler* da carteira e dos seus *listeners*.

3.2.13.1 Data Handler

- **PortfolioDataHandler** – O *portfolioDataHandler* mantém em memória um conjunto de *clientIds*. A cada *clientId* está associado um objeto *clientPortfolio*. O par *clientId – clientPortfolio* forma cada entrada da estrutura de dados *hash map* mantida e gerida pelo *portfolioDataHandler*. A *hash map* é definida do seguinte modo:

  ```
  data = {clientId1: clientPortfolio1,...,clientIdN:clientPortfolioN}
  ```

- **ClientPortfolio** – O objeto *clientPortfolio* representa e armazena toda a informação referente à carteira de um cliente, tendo, por isso, como variável de instância, um objeto do tipo *ClientPortfolioData* que estende uma estrutura de dados do tipo *hash map*. Em seguida, é exemplificada uma entrada de dados na *hash*, representada pelo par chave-valor.

  ```
  data = {portfolioKey1:portfolioEntry1,...,portfolioKeyN:portfolioEntryN}
  ```

  A *portfolioKey* é uma string única formada pela praça, mercado, *symbol* e *ISIN* do título. Por exemplo, um título em carteira da praça *XLIS*, mercado *CS*, *symbol* BCP e *ISIN* PTBCP0AM0007, forma uma chave de entrada na *hash* do género:

  ```
  Chave: ‘‘XLIS+CS+BCP+PTBCP0AM0007’’ -> Valor: portfolioEntry
  ```

  À chave *portfolioKey* está associado um objeto *portfolioEntry* que contém toda a informação do título que representa uma entrada na carteira do cliente. Portanto, o objeto estende uma *hash map* do tipo:

  ```
  Chave: ‘‘405’’ -> Valor: ‘‘BCP’’
  Chave: ’’406’’ -> Valor: ’’4 000 000’’
  ```

  A Chave “405”, por exemplo, corresponde ao nome do título e a chave “406” representa a quantidade em carteira do título em questão. O objeto *clientPortfolio* também mantém e gere uma lista de *listeners*.

- **Incoming Messages**

  - **Update** – A mensagem de *update* que chega ao objeto *model portfolioDataHandler* transporta toda a informação dos títulos pertencentes à carteira de um cliente. O objeto *model* vai sendo notificado com mensagens *update* sempre que necessário, ou seja, sempre que houver alterações na carteira do cliente, seja pela entrada de um título, seja pela venda de um certo número de ações de um dos título em carteira, qualquer que seja a alteração. A primeira mensagem de *update* acarreta toda a informação da carteira do cliente.
  - **Add Listener** – Mensagem recebida pelo objeto *model portfolioDataHandler*, a fim de subscrever um novo objeto *view* junto da carteira de um cliente. Portanto, no envio da mensagem *Add Listener* é enviada uma *string* com a identificação do cliente (*clientId*) de modo a que se obtenha o objeto *clientPortfolio* correto. Depois, o novo *listener*, já adicionado ao *array* de *listeners* da carteira, passa a ser notificado das alterações de dados da carteira do cliente que subscreveu.
  - **Remove Listener** – É a lógica inversa da mensagem *Add Listener*. O objeto *view* que enviou esta mensagem é removido do *array* de *listeners* da carteira, deixando de ser notificado de futuras alterações dos dados da carteira.
  - **Connection Status** – É uma mensagem que se forma aquando da mudança do estado da ligação da aplicação.
  - **Notify On Error** – Mensagem enviada pelo servidor ao *portfolioDataHandler*, a fim de notificar a ocorrência de um erro. O erro pode ocorrer por diversas razões, por exemplo, o servidor ao tentar carregar a carteira de um cliente que não exista na base de dados.

- **Outgoing Messages**

  - **Portfolio Data Request** – Mensagem de pedido de carteira de um cliente ao servidor. Na mensagem é enviado o *clientId* associado ao cliente. Esta é uma mensagem do tipo *subscribe*.
  - **Portfolio Data Unrequest** – Mensagem de *unsubscribe* da carteira de um cliente. O objeto *portfolioDataHandler* deixa de receber mensagens de *updade* relativas à carteira do *clientId*, transmitido na mensagem como argumento.
  - **Notify On Portfolio Error** – Mensagem enviada a todos os *listeners* de uma carteira, a fim de serem notificados da ocorrência de um erro.

Capítulo 4

Utilização

Neste capítulo é apresentada a constituição de cada componente da interface gráfica, para *iPhone* e para *iPad*, o seu funcionamento e respetiva navegação, agrupados por modos de operação. Serão também aqui descritos alguns elementos gráficos que fazem parte da *API* interface disponibilizada pelo sistema operativo *iPhone Operating System (iOS)* 5 da *Apple*. Todos os componentes gráficos e a sua organização foram implementados durante os nove meses de período de estágio curricular.

4.1 Dados de Mercado

A interface Dados de Mercado permite ao utilizador visualizar a informação sobre uma lista de títulos de mercado, agrupados por *tabs*[1], que podem ser mantidas e geridas pelo utilizador. Este pode efetuar operações de, adição, remoção e ordenação sobre cada *tab* do elemento gráfico que as contém. Normalmente, a cada *tab* é atribuído o nome de um índice (*PSI20*) ou mercado (*NASDAQ*), ficando ao critério do utilizador. A figura 4.1 ilustra este elemento gráfico denominado DMTabMenu.

Figura 4.1: Elemento gráfico DMTabMenu.

O *DMTabMenu* estende um elemento gráfico (*UIScrollView*) do sistema *iOS 5* da *Apple*. Segundo os princípios da *Apple*, cada modo de operação deve ter um título,

[1]Conjunto de separadores ao longo de uma barra que permitem a navegação em janelas distintas.

que por sua vez está associado a um elemento gráfico, chamado de *navigation bar* e que pode ser visualizado na figura 4.2.

Figura 4.2: Elemento gráfico Navigation Bar.

O *navigation* bar permite a navegação através de uma hierarquia de informação e que, opcionalmente, gere o conteúdo da interface gráfica. Está contido num controlador de navegação (*Navigation Controller*) que é um objeto que gere a exibição de uma hierarquia de intefaces gráficas (*views*) personalizadas.

O utilizador inicialmente consegue visualizar, no caso do *iPhone*, apenas duas informações de cada título, a cotação e a oscilação percentual. Isto acontece devido ao tamanho reduzido do ecrã do *iPhone*, que no caso do *iPad* é ultrapassado pelas dimensões que possui. No *iPhone*, a solução passou por criar um novo elemento gráfico, denominado *DMSubBar*(figura 4.3), que permitisse a troca de colunas.

Figura 4.3: Elemento gráfico DMSubBar.

Portanto, ao clicar na coluna da cotação o valor muda e o utilizador passa agora a visualizar informação sobre o *Ask Price* e a oscilação percentual dos títulos.

Um outro elemento gráfico muito importante e que permite a navegação por toda a aplicação, é o *tab bar*[1]. Este é uma barra horizontal que contém itens, representados por um título e uma imagem. De entre os itens, apenas um pode estar selecionado em qualquer estado da aplicação. O *tab bar* é um objeto gráfico independente, mas que normalmente é utilizado em conjunto com um *tab bar controller* que gere a interface de seleção dos diversos modos de operação. A figura 4.4 apresenta o elemento tab bar.

Figura 4.4: Elemento gráfico Tab Bar.

[1]É um elemento gráfico implementado pela Apple que permite a navegação através de diversos modos de operação.

A cada *tab* do *tab bar controller* está associado um *view controller*. Quando o utilizador seleciona uma *tab* específica, o *tab bar controller* mostra a *root view* do *view controller*, substituindo as *views* anteriores.

O *tab bar controller* apresenta os vários modos de navegação disponíveis no SifoxDeal Mobile. São eles, os dados de mercado (mercados), livro de ordens (ordens), negociar, carteira e financeiro. Na figura 4.5, é apresentada a interface gráfica dos dados de mercado para *iPhone* e *iPad*, respetivamente:

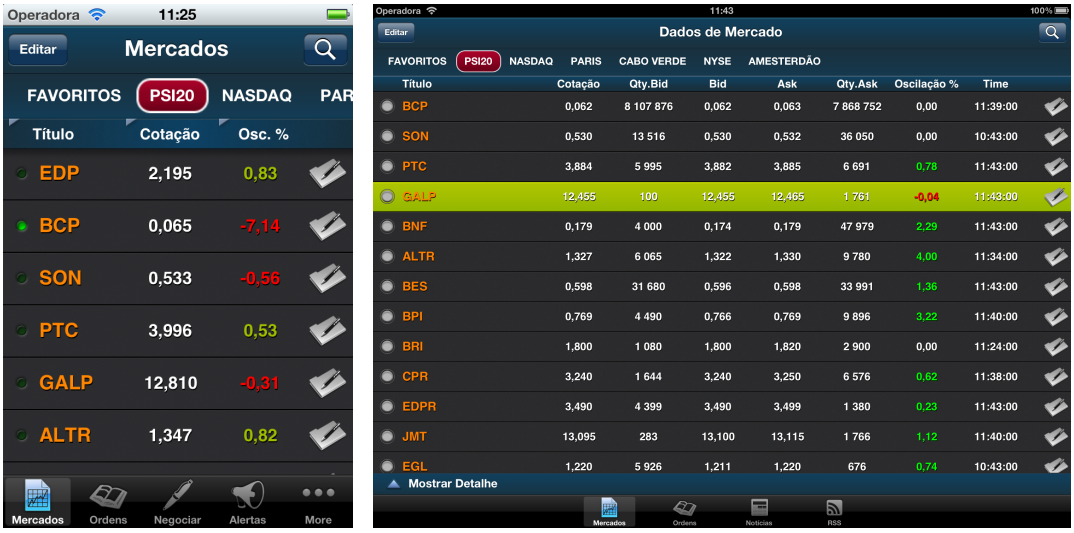

Figura 4.5: Ecrã de dados de mercado. Do lado esquerdo referente ao iPhone e do lado direito ao iPad.

O utilizador, ainda na interface de dados de mercado, pode clicar no botão editar presente no *navigation bar*, e modificar (eliminar ou ordenar) as linhas ou células de títulos de uma *tab* tal como ilustra a figura 4.6.

Para a pesquisa e inserção de títulos, está disponível um botão *search*, presente no *navigation bar*, que o leva a navegar até à interface gráfica de pesquisa (dicionário), onde serão então encontradas as opções de pesquisa e inserção de títulos no ecrã de dados de mercado.

Por fim, a partir dos dados de mercado, o utilizador pode ter acesso ao detalhe de um título, clicando sobre uma célula da lista títulos. Por exemplo, se o utilizador selecionar a *tab NASDAQ*, referente aos mercados americanos, e depois selecionar o título *FB*, a aplicação levará o utilizador ao ecrã de detalhe do título, tal como ilustra a figura 4.7.

Nesta interface, é apresentada toda a informação específica a um título de mercado,

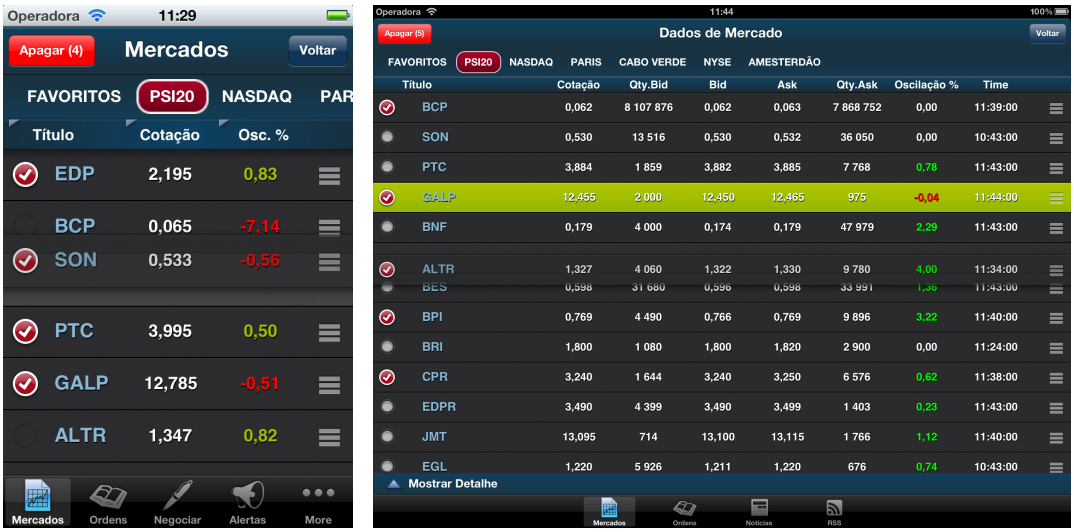

Figura 4.6: Ecrã de dados de mercado em modo de edição.

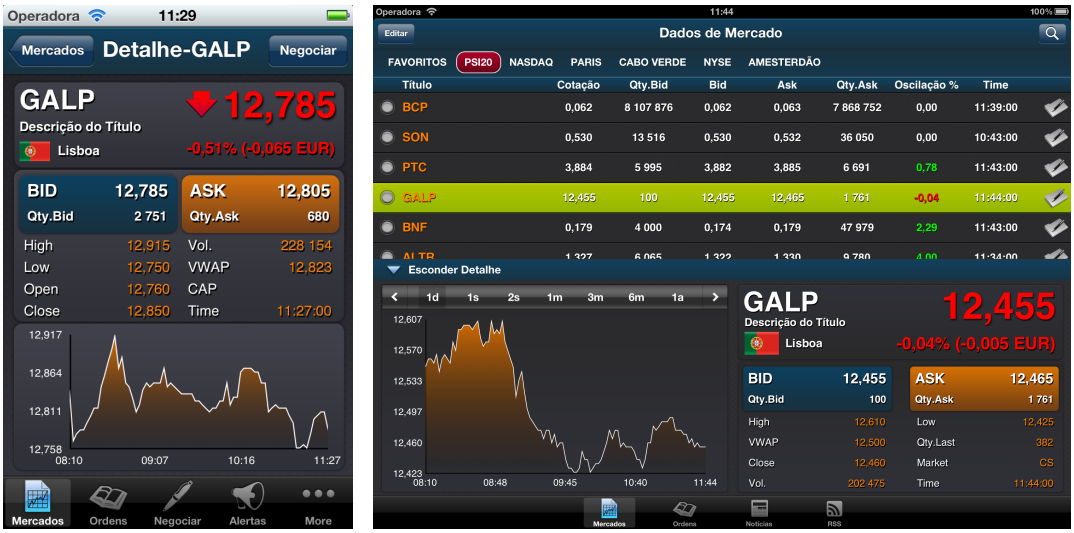

Figura 4.7: Ecrã de detalhe de um título.

inclusive um gráfico de *intraday* que é atualizado em *real time* aquando da realização de negócios (venda e compra de ações do respetivo título) e consequente mudança da cotação. A partir deste ecrã, também é possível ao utilizador navegar até ao modo de operação negociar.

4.2 Livro de Ordens

A interface Livro de Ordens (figura 4.8) permite ao utilizador visualizar a informação de uma lista de ordens agrupadas por cliente (*clientId*).

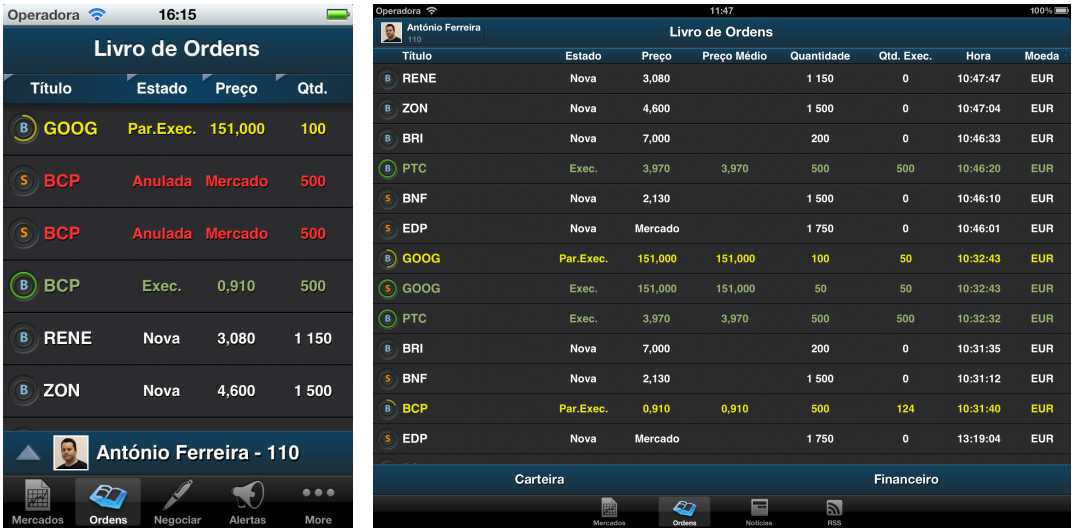

Figura 4.8: Ecrã do Livro de Ordens.

Esta interface gráfica do livro de ordens é, de certa forma, semelhante à dos dados de mercado. Contém um *navigation bar*, um *DMSubBar* e um novo elemento gráfico, chamado *DMClientChooser*.

O *navigation bar*, neste caso, só apresenta o título do ecrã. O *DMSubBar* possui as mesmas características que o dos dados de mercado, mas com três colunas, mostrando informações diferentes. Neste ecrã, é de salientar o ícone que se apresenta no início de cada linha que representa uma ordem. Um ícone desenhado com o objetivo de representar o tipo de ordem B (*Buy*) ou S (*Sell*), compra ou venda respetivamente, e a percentagem da quantidade executada.

O *DMClientChooser* (figura 4.9), que também foi desenvolvido durante o estágio, tem como função disponibilizar ao utilizador uma lista de clientes. O utilizador pode escolher qualquer *clientId* e, posteriormente, efetuar operações e consultar informações sobre o mesmo.

Portanto, pela utilização do *DMClientChooser*, é possível ver o livro de ordens de vários clientes. Este elemento gráfico permite também uma pesquisa em *live search* de clientes, porque um utilizador, na condição de *trader* ou não, pode ter um número de clientes na ordem dos milhares.

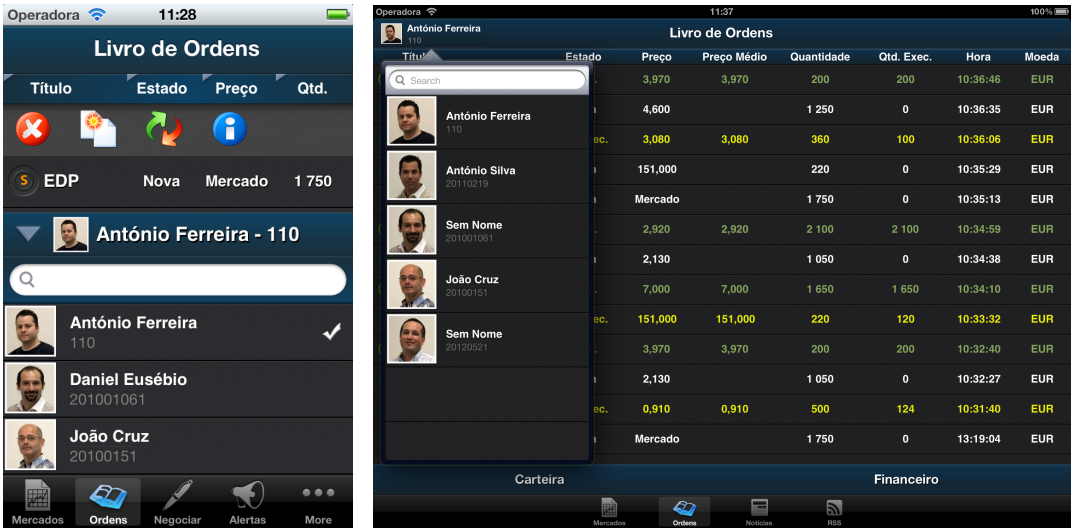

Figura 4.9: O elemento gráfico ClientChooser no ecrã do livro de ordens.

Por fim, o utilizador pode lançar uma célula para a esquerda ou para a direita *"para fora do ecrã"* e obter algumas opções sobre a ordem. As opções disponíveis são a anulação da ordem, duplicação da ordem, modificação da ordem e informação da ordem.

4.3 Negociar

A interface Negociar permite ao utilizador o envio de ordens para os mercados financeiros. O ecrã de negociar, tal como nos outros modos de operação já descritos, possui um *navigation bar* com o título do ecrã. Apresenta também duas opções, voltar para trás e enviar ordem, no seu lado esquerdo e no seu lado direito, respetivamente.

O *DMClientChooser* pode também ser encontrado na interface negociar para que o utilizador possa enviar ordens referentes a um *clientId* específico. No *iPad*, a disposição do *DMClientChooser* muda, mas a lógica mantém-se. A interface Negociar é apresentada pela figura 4.10.

A interface gráfica do negociar no *iPhone* é semelhante à do *iPad*. Contudo, no caso do segundo, o ecrã de negociar é apresentado como um *modal view controller* que permite interromper o fluxo atual da aplicação, mostrando um novo conjunto de *views*. É frequentemente utilizado quando se pretende mostrar algo temporário ou, então, quando se pretende implementar interfaces alternativas.

O utilizador para negociar necessita de fornecer à interface informações, como tipo,

Figura 4.10: Ecrã do Negociar.

quantidade, preço e data de validade, que são de carácter obrigatório para o sucesso do envio da ordem. Só o campo preço stop da interface no *iPad* é de caráter opcional.

Como no mundo dos mercados se pode ganhar e perder capital em questão de segundos, é importante que o envio de uma ordem seja rápido, assim como o preenchimento dos campos. Por isso, a interface foi desenhada com três botões, *Last*[1], *Bid* e *Ask*, não só para fornecer informações ao utilizador dos preços do vendedor e do comprador, mas também para que possa, ao clicar num deles, preencher o campo do preço automaticamente. O mesmo acontece com o campo da quantidade, mas neste caso foi implementada uma barra adicional que se encontra sobre o teclado com quantidades predefinidas.

4.4 Dicionário

A interface Dicionário (pesquisa figura 4.11) permite ao utilizador a pesquisa de títulos por nome ou pela descrição. O *navigation bar* deste ecrã é definido pelo título, o botão de voltar para atrás e um outro botão inserir do lado direito.

A interface apresenta um elemento gráfico diferente de todos os outros implementados e que podemos encontrar nos modos de operação anteriores, o elemento gráfico *Search Bar*. O *Search Bar* implementa um campo de texto, um botão cancelar e um botão de pesquisa. Bem, na realidade não é o *Search Bar* que efetua a pesquisa, mas sim

[1]Último preço do título em questão.

um objeto que esteja de acordo com o protocolo *UISearchBarDelegate*, que neste caso é o *dictionaryViewController* (instância da interface dicionário). O *dictionaryViewController* tem então a responsabilidade de implementar as ações quando o texto é inserido e quando os botões são clicados.

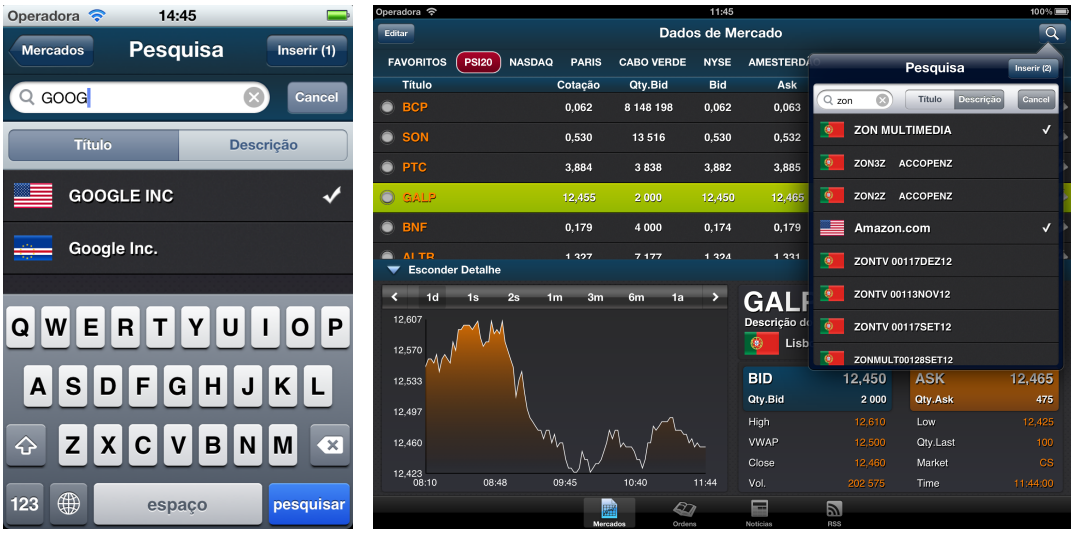

Figura 4.11: Ecrã do Dicionário.

O utilizador pode navegar até à interface gráfica do dicionário de duas maneiras e com objetivos diferentes. A partir dos dados de mercado, clicando no ícone *search* no *navigation bar*, com o objetivo de inserir um novo título na interface de dados de mercado, ou a partir do ecrã negociar quando se pretende escolher um título para qual se pretende enviar uma ordem. Esta escolha é possível clicando sobre o nome do título atual na interface de negociação.

Em relação ao *iPad*, o ecrã de pesquisa tem um aspeto e um formato um pouco diferente da do *iPhone*. Isto, mais uma vez, pela razão do ecrã do *iPad* ser maior. Não era necessário o preenchimento de todo o ecrã para uma simples pesquisa temporária. Implementou-se então um controlador *UIPopoverController* que é usado para gerir o conteúdo no *popover*. Os *popovers* são usados para mostrar informação temporária e permanecem visíveis até que o utilizador toque fora da área coberta pelo mesmo.

4.5 Carteira

A interface gráfica da Carteira (figura 4.12) permite ao utilizador visualizar os títulos em carteira de um determinado cliente.

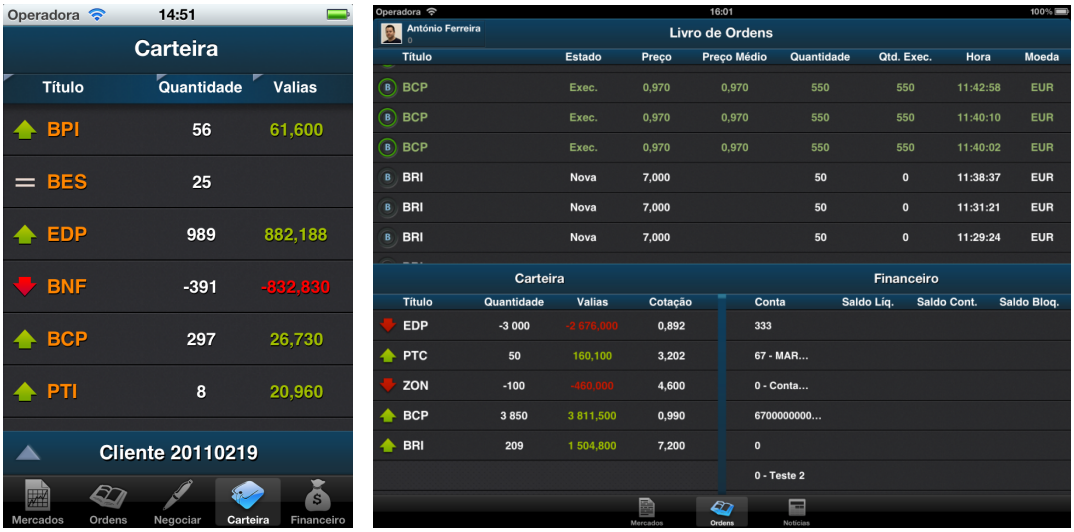

Figura 4.12: Ecrã da Carteira.

O utilizador pode consultar diversas informações sobre os títulos em carteira pela mudança das colunas através do *DMSubBar*. Informações como a quantidade de ações do título, as valias, a cotação atual, entre muitas outras.

No *iPad* optou-se por implementar uma *slide view*, inserida no livro de ordens, contendo a carteira e o financeiro. Esta *view* pode ser escondida sempre que o utilizador assim o entender, bastando um simples clique sobre a sua barra. Portanto, no *iPad*, sempre que o utilizador mudar de cliente, verá não só as ordens do cliente que selecionou, mas também a carteira e o financeiro.

O utilizador pode ainda negociar partes ou a totalidade dos títulos de um cliente em carteira. Para o devido efeito, basta clicar sobre a linha ou célula do título de uma carteira que a aplicação levará o utilizador até ao ecrã negociar.

4.6 Financeiro

A interface do Financeiro permite ao utilizador visualizar a informação das contas bancárias pertencentes a um cliente, que pode ter mais que uma conta bancária. A partir da navegação no *DMSubBar*, o utilizador tem acesso a todas as informações disponíveis das contas, como saldo líquido, moeda, saldo a liquidar, saldo contabilístico, saldo bloqueado, entre outras. A figura 4.13 apresenta a interface do financeiro.

Tal como no modo de operação carteira, no *iPad*, a interface do financeiro encontra-se

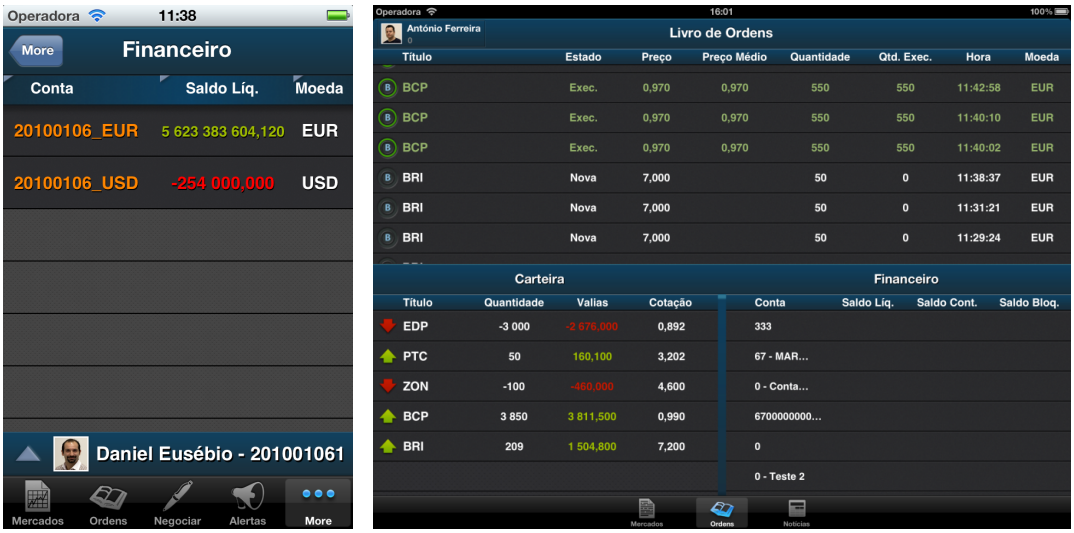

Figura 4.13: Ecrã do Financeiro.

inserida numa *slide view*, no ecrã do livro de ordens.

Capítulo 5

Mobilidade

Este capítulo apresenta o modelo de mobilidade utilizado, assim como os protocolos de encriptação e respetivos algoritmos utilizados na comunicação, entre o SifoxDeal Mobile e o ServerDeal Web.

5.1 O modelo cliente-servidor

O modelo cliente-servidor é um modelo arquitetural dividido em duas partes, a parte cliente e a parte servidor, que normalmente comunicam entre si através de uma rede. É um paradigma clássico em sistemas distribuídos, utilizado pela maioria das aplicações e comunicações de rede. Segundo este paradigma, o servidor é visto como um elemento ouvinte que espera passivamente por um pedido vindo da parte cliente, com o objetivo de obter recursos de rede. Após o envio do pedido por parte do cliente, é função do servidor proceder ao seu processamento e enviar o resultado ao cliente que aguarda a resposta [14].

Tipicamente, a parte cliente é uma máquina que corre software que necessita de recursos, disponíveis algures numa máquina remota chamada de servidor, que também corre software, mas que neste caso tem a responsabilidade de manter e gerir a informação e serviços necessários aos clientes. O servidor pode interagir com um grande número de clientes ao mesmo tempo, assim como ligar-se a outros servidores de modo a obter recursos necessários a um pedido específico. O cliente, por sua vez, pode conectar-se a mais do que um servidor, de uma só vez, e permitir a interação com o utilizador final. A figura 5.1 ilustra de uma forma genérica o paradigma cliente-servidor.

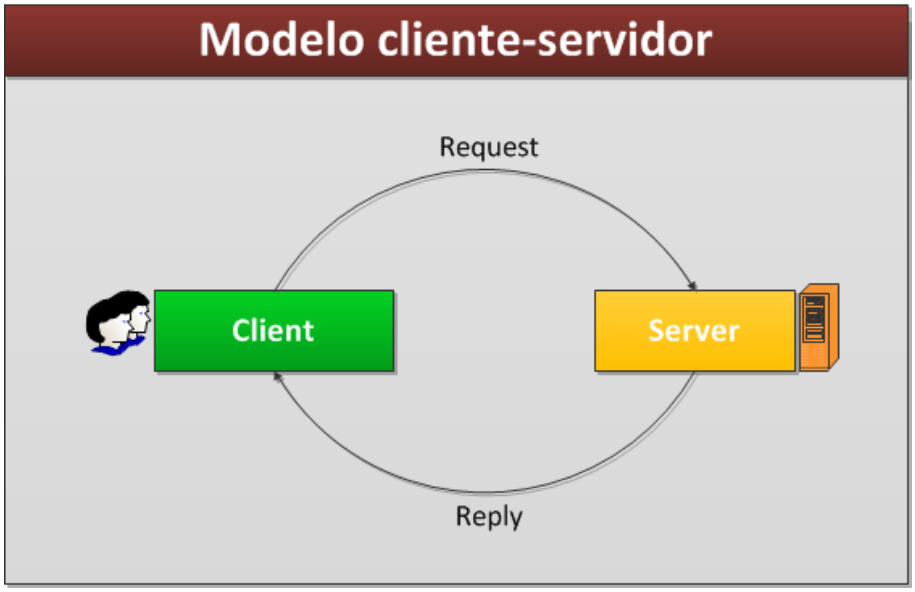

Figura 5.1: Modelo cliente-servidor.

Este modelo oferece uma série de vantagens, tais como:

- Informação centralizada;
- Facilidade de manutenção;
- Facilidade de gestão e segurança da informação;
- Suporte a clientes diferentes.

Como qualquer sistema, para além das vantagens, também possui algumas desvantagens, tais como:

- O servidor pode ser sujeito a *overhead* se receber uma quantidade de pedidos simultâneos, por parte dos clientes, superior à que pode suportar;
- No caso de falha do servidor, os clientes deixam de poder realizar pedidos e receber as respetivas respostas - é uma desvantagem em relação ao modelo *P2P*.

O modelo cliente-servidor está subjacente na arquitetura do SifoxDeal Mobile, em que o servidor é o ServerDeal Web e o cliente é um dispositivo móvel *iOS* com o SifoxDeal Mobile instalado, tal como ilustra a figura 5.2.

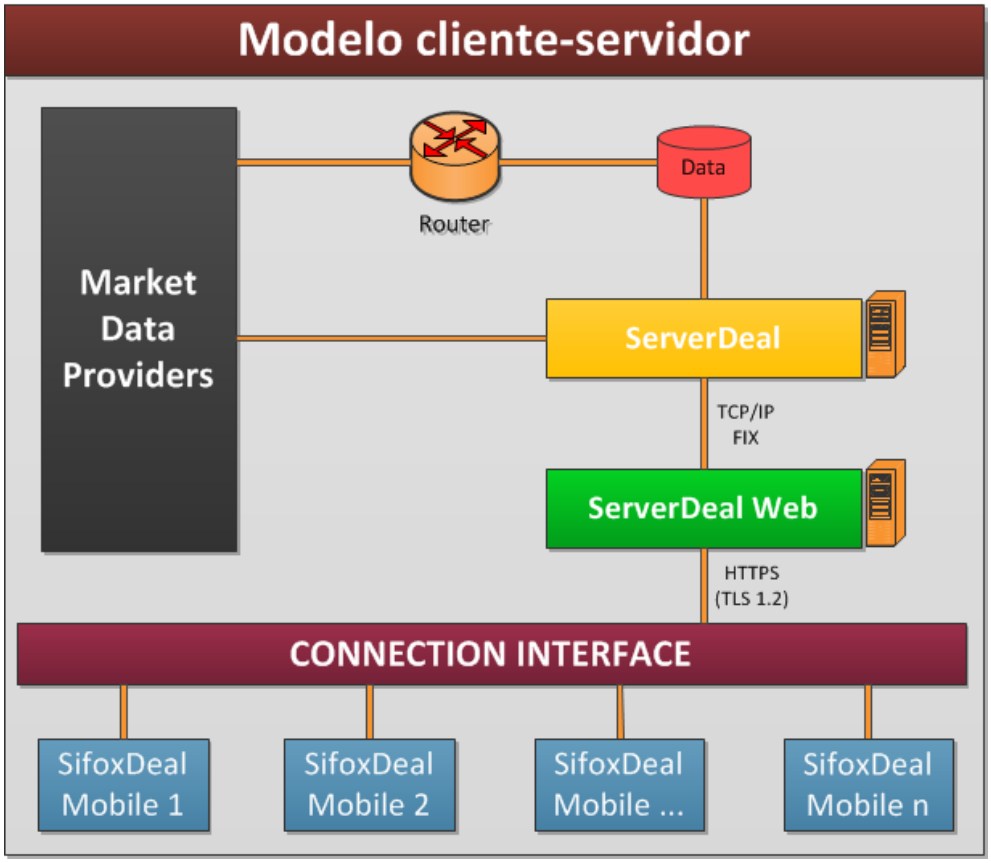

Figura 5.2: Modelo cliente-servidor SifoxDeal Mobile.

A adoção deste modelo pela arquitetura do SifoxDeal Mobile deve-se essencialmente a três caraterísticas, são elas: escalabilidade, integridade e mobilidade.

A escalabilidade, porque o sistema cliente-servidor da arquitetura SifoxDeal Mobile pode evoluir facilmente para um sistema mais denso pela adição de novos dispositivos móveis *iOS* que correm a aplicação.

Relativamente à integridade, a caraterística mais importante, pelo fato de se tratar de uma aplicação de negociação em bolsa que envolve operações críticas como transações de grandes volumes de capital, mantém a informação, dados e código, de forma centralizada, facilitando a manutenção e integridade da informação.

A mobilidade, porque o servidor, como processo, pode estar alojado em qualquer ponto da rede; pode, por exemplo, estar inserido na mesma máquina do processo cliente ou alojado numa máquina remota, algures na rede, a milhares de km de distância do processo cliente.

Outros aspetos importantes são o agrupamento de serviços e recursos partilhados.

Pela troca de mensagens, o servidor recebe um pedido de requisição de um serviço que é implementado e mantido pelo servidor. Os serviços podem ser alvos de atualizações sem que os clientes sejam afetados. A respeito da partilha de recursos, e pelo fato de existir uma relação de um-para-muitos entre servidor e clientes, é função do servidor servir vários clientes ao mesmo tempo e gerir os acessos aos recursos partilhados [14].

5.2 Comunicação

O SifoxDeal Mobile comunica com o ServerDeal Web através de uma infra-estrutura standard em *HTTPS*. O servidor está integrado com o *IIS* que é um servidor *web* desenvolvido pela Microsoft que suporta protocolos aplicacionais como *HTTP*, *HTTPS*, *FTP*, *FTPS*, *SMTP* e *NNTP*.

O protocolo *HTTPS*[18] é uma extensão ao protocolo *HTTP*[5] que adiciona uma camada de segurança de nível inferior que utiliza o protocolo *SSL/TLS*[19]. Basicamente, esta camada permite o estabelecimento de um canal de comunicação seguro pela utilização de protocolos criptográficos e certificados digitais.

As mensagens trocadas entre o terminal SifoxDeal Mobile e o servidor, que fazem chegar ao utilizador final toda a informação dos mercados financeiros nacionais e internacionais e que permitem a negociação, são críticas e suscetíveis de serem intercetadas por indivíduos mal intencionados. Exatamente por esta razão todo e qualquer pedido/resposta *HTTP* entre o *mobile* e o ServerDeal Web é encriptado utilizando os protocolos *SSL/TLS* sobre uma comunicação *TCP* orientada a sockets[1].

O *TCP* e o *UDP* são protocolos de transporte do modelo *OSI* que correm por cima da camada de rede *IP*. O *TCP* é orientado à conexão, ou seja, antes de existir qualquer troca de informação, é estabelecido e acordado previamente um canal de comunicação entre os dois terminais envolventes. Este protocolo garante a entrega ordenada dos pacotes, assim como a sua retransmissão perante uma perda eventual.

Comparativamente ao *UDP*, é claramente um protocolo mais complexo e lento na transferência de dados, exatamente porque fornece este tipo de serviços, de entrega ordenada e retransmissão de pacotes perdidos. Assim, compreende-se que o *TCP* é utilizado em situações em que a integridade da mensagem é mais importante do que o tempo de transmissão, enquanto que o *UDP* é utilizado em casos em que a velocidade

[1] Uma socket é a extremidade de um processo de comunicação entre duas aplicações através de uma rede de computadores.

de transmissão é o mais importante [14].

O protocolo *SSL/TLS* tem como principal objetivo proporcionar privacidade e integridade dos dados na comunicação entre o terminal SifoxDeal Mobile e o ServerDeal Web. É composto por duas camadas, a *TLS Record Protocol* e a *TLS Handshake Protocol*. O primeiro protocolo proporciona segurança a nível do canal de comunicação – a comunicação é privada e confiável. A privacidade é obtida através da encriptação dos dados pela utilização de criptografia simétrica (*AES*, *RC4*, entre outros). Esta é baseada na partilha de um segredo (uma chave simétrica) que somente ambas as entidades envolventes devem ter conhecimento. Contudo, a criptografia simétrica tem um problema associado, no ato da partilha do segredo (antes da transferência de qualquer bit de dados) a comunicação pode ser intercetada por terceiros que podem usufruir do segredo que agora é partilhado por três entidades. Portanto, de modo a ultrapassar este problema, por cada comunicação que é estabelecida são geradas chaves para a encriptação simétrica, baseadas na negociação de um segredo assegurada pelo protocolo *TLS Handshake* [19].

A integridade dos dados é mantida pela utilização de *Message Authentication Codes* (MAC)[12] que usufruem de funções *hash* como o *SHA-1*.

É então função do protocolo *TLS Handshake* permitir a autenticação de ambas as entidades para que possam "negociar" um algoritmo de encriptação, assim como as chaves criptográficas, antes de ser transmitida qualquer tipo de informação entre os envolventes. A identidade de ambos, SifoxDeal Mobile e ServerDeal Web, são autenticadas utilizando criptografia assimétrica ou criptografia de chave pública.

Finalmente, é estabelecido um canal de comunicação entre o cliente e o servidor seguro e confiável.

Capítulo 6

Tecnologias e Ferramentas Utilizadas

Este capítulo descreve as tecnologias e ferramentas que foram utilizadas ao longo do desenvolvimento do SifoxDeal Mobile.

6.1 Xcode

O *Xcode* é um ambiente de desenvolvimento integrado criado pela *Apple* para o desenvolvimento de aplicações *iOS* e *Mac*. A interface do *Xcode* (figura 6.1) integra uma área de implementação e edição de código, de desenho de interfaces de utilizador com o *Interface Builder* de que dispõe, um compilador *Apple LLVM* totalmente integrado e todo o tipo de testes e *debugging*, tudo numa simples janela. É uma espécie de ecossistema para os programadores.

Inclui um conjunto de ferramentas importantes e essenciais ao desenvolvimento. Uma ferramenta para análise de performance e comportamento, um simulador (*iOS Simulator*) capaz de executar uma aplicação, tal como um dispositivo *iOS* real, e mais recentemente o *SDK iOS* e *Mac OS X*.

O Xcode foi o ambiente de desenvolvimento utilizado para o desenvolvimento do SifoxDeal Mobile que permitiu a edição e desenvolvimento do código, assim como a análise de performance, simulação e *debug* da aplicação.

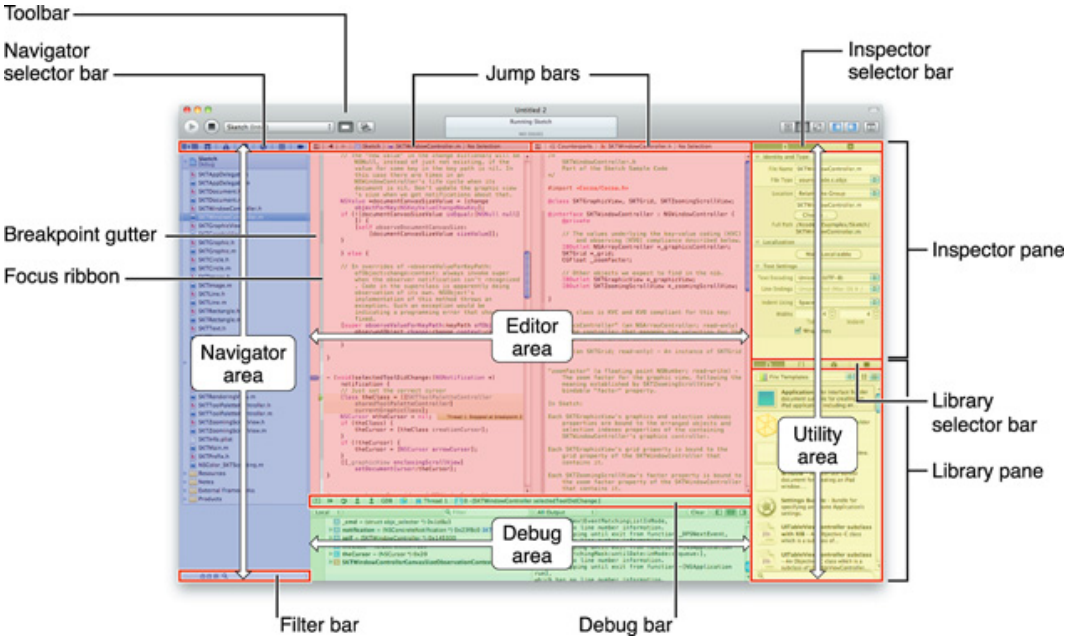

Figura 6.1: Interface do Xcode.

6.1.1 Apple LLVM compiler

O compilador *Apple LLVM*, baseado no projeto *open source LLVM.org*, faz parte da próxima geração da tecnologia de compilação da *Apple*.

Este compilador é rápido. Compila duas vezes mais rápido que o compilador convencional *GCC*. Foi construído do zero como um conjunto de bibliotecas altamente otimizadas e fáceis de estender, concebidas para as arquiteturas atuais dos processadores modernos [15]. Uma imagem ilustrativa do *llvm* em ação é apresentada pela figura 6.2.

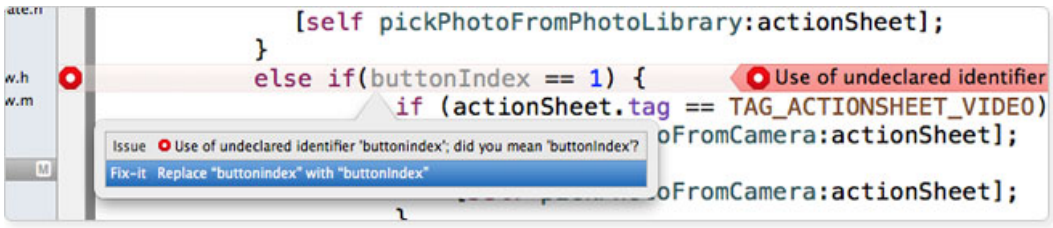

Figura 6.2: Apple LLVM compiler.

Notificações de sintaxe e sugestões *code completions* são tarefas do *Apple LLVM* que, em *background*, avaliam constantemente o que é escrito.

6.1.2 Instrumentos de análise de performance e comportamento

As aplicações, para além de um bom design intuitivo e estruturado, devem ser rápidas. Os instrumentos de análise de performance e comportamento (figura 6.3) disponibilizam a possibilidade de um estudo pormenorizado sobre a performance de uma aplicação pela recolha de informações como, disco, memória e utilização de *CPU* em *real time*. Os dados recolhidos são mostrados graficamente como faixas ao longo do tempo, tornando-se fácil encontrar zonas problemáticas.

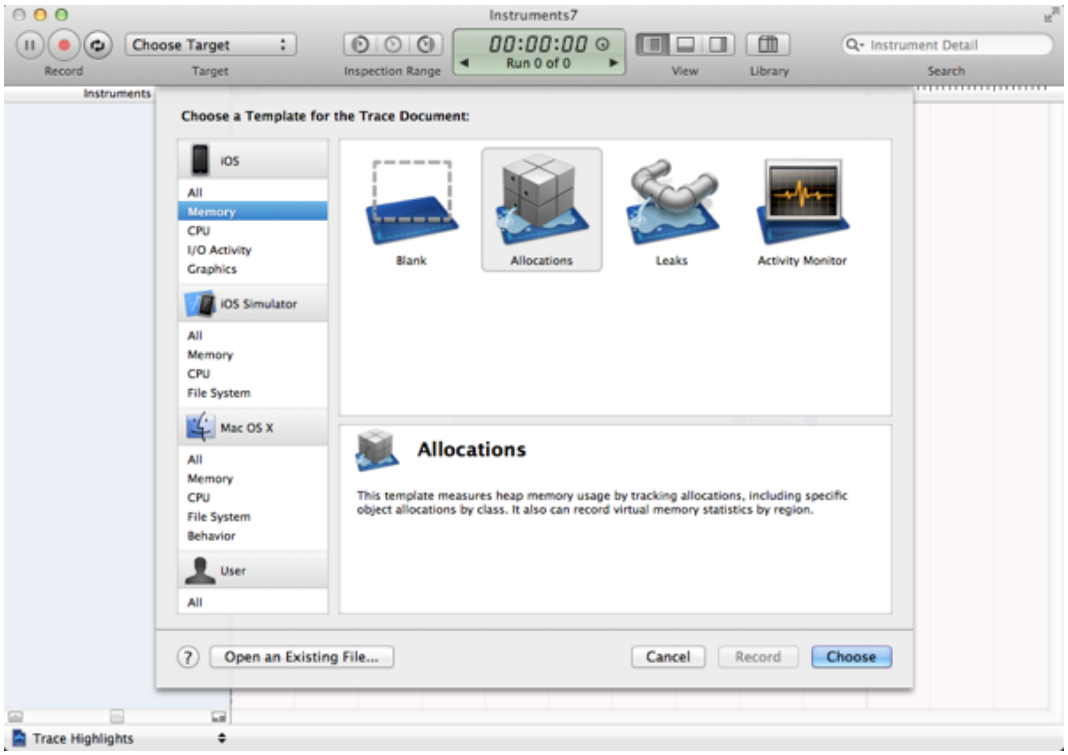

Figura 6.3: Instrumentos de análise de performance e comportamento.

Com estes instrumentos, o utilizador pode juntar o útil ao agradável, bom *design* e performance.

6.1.3 iOS Simulator

O *iOS Simulator* (figura 6.4) é uma ferramenta importante no desenvolvimento do SifoxDeal Mobile. Executa uma aplicação como de um *device iOS* real se tratasse. Isto porque é rápido, permite testar a interface do utilizador e encontrar problemas durante o seu desenho. Permite, ainda, o *debugging* e suporta a maioria das funcionalidades e opções de um dispositivo real. É também útil pelo fato de poder ser utilizado pela

ferramenta de análise de performance e comportamento, a fim de, principalmente, se obter a percentagem de memória usada e posterior análise antes da instalação em dispositivos físicos *iOS*.

Figura 6.4: iOS Simulator.

Pode simular quatro dispositivos (*iPhone*, *iPhone* retina, *iPad* e *iPad* retina) e várias versões do *iOS*.

6.2 O Sistema iOS

O *iOS* é um sistema operativo móvel desenvolvido e distribuído pela *Apple Inc.* Foi construído com base no *OS X* e simplificado para ser compacto e eficiente, tirando o máximo partido do *hardware* dos dispositivos móveis *iOS*. Lançado ao público em 2007 para o *iPhone* e *iPod touch*, era conhecido como *iPhone OS*. Mais tarde, foi estendido e melhorado, suportando novos dispositivos como o *iPad* e a *Apple TV*. Este sistema operativo, contrariamente ao da principal concorrente *Google*, não está disponível para dispositivos com *hardware* diferenciado.

A interface de utilizador do *iOS* e a sua navegação são caracterizadas pela interação

direta entre o utilizador e o dispositivo. São utilizados gestos multi toque como apenas tocar no ecrã, deslizar o dedo, rotação do dedo e movimento de pinça, todos com especificas definições no contexto do sistema operativo *iOS* e a sua interface multi toque.

As caraterísticas do sistema *iOS* são agrupadas em quatro camadas abstratas. São elas, *Cocoa Touch*, *Media*, *Core Services* e *Core OS*. O conjunto formado por estas quatro camadas produz uma espécie de super camada intermédia entre a aplicação e o *hardware*. As duas camadas mais inferiores, a camada *Core Services* e *Core OS*, são camadas de baixo nível, desenvolvidas em C (linguagem C). As outras duas camadas, *Cocoa Touch* e *Media*, são camadas mais próximas da aplicação que fornecem *frameworks* mais avançadas escritas em C e Objective-C. As camadas são descritas de forma breve a seguir.

6.2.1 Cocoa Touch

A camada *Cocoa Touch* é a camada mais utilizada no desenvolvimento de aplicações *iOS*. Por isso, é constituída pelas principais bibliotecas para a criação de aplicações como a *Foundation framework* e a *UIKit*. A *Foundation framework* representa os tipos básicos de dados e a *UIKit* fornece os componentes gráficos, essenciais à construção da interface do utilizador.

Os principais serviços geridos por esta camada são o reconhecimento de gestos, eventos, notificações globais e remotas, acesso aos sensores, câmara, funcionalidades multi-tarefa, e muitas outras.

6.2.2 Mídia

Na camada Mídia são encontradas as bibliotecas que manipulam os recursos gráficos, de áudio e vídeo. Todas as operações sobre estes recursos, como, a gravação de áudio e vídeo, animação e renderização 2D e 3D, são realizadas nesta camada.

6.2.3 Core Services

Esta camada disponibiliza serviços fundamentais utilizados por todas as aplicações. Embora não seja utilizada, normalmente, de uma forma direta, é uma camada extre-

mamente importante, pelo que algumas partes do sistema são constituídas com base nestes serviços.

Os principais serviços desta camada são os serviços de telefonia, *iCloud*[1], preferências, livro de endereço, rede, entre muitos outros.

6.2.4 Core OS

A camada *Core OS* é a camada com mais baixo nível de abstração, muito próxima do *kernel*. Fornece acesso direto a funcionalidades de baixo nível, como *sockets*, certificados, gestão de energia, gestão de bateria, gestão de memória, segurança, entre outros.

6.3 Gand Central Dispatch (GCD)

O GCD é uma tecnologia desenvolvida pela *Apple Inc.* que oferece suporte à execução de código concorrente em *hardware multi core* no *iOS* e *OS X*. Foi introduzida inicialmente pela *Apple*, no *Mac OS X 10.6*, e hoje encontra-se presente também no *iOS 4* e superiores. O principal objetivo desta tecnologia é de tornar o uso de *threads* e mecanismos de sincronização tradicionais mais transparentes ao programador e, ao mesmo tempo, possibilitar a execução de tarefas em paralelo, de uma forma mais simples e eficiente.

O GCD mantém e gere uma fila FIFO, para a qual as aplicações podem submeter tarefas inseridas num contexto *block*[2]. Um *block* é uma extensão da linguagem C, muito semelhante a uma função C, que agrupa um conjunto de instruções que pode ser passado como argumento e executado noutra ocasião. Posteriormente, é função do GCD distribuir as tarefas por um conjunto de *threads* (*thread pools*) que são geridas automaticamente pelo sistema.

O GCD oferece três tipos de filas:

- ***Main*** – as tarefas são executadas de forma sequencial pela *thread* principal da

[1]É uma tecnológica desenvolvida pela Apple que permite o armazenamento da informação dos utilizadores na Internet.

[2]Um block é semelhante a uma função C que contém um conjunto de instruções que pode ser executado noutra ocasião e de forma concorrente.

aplicação.

- ***Concurrent*** – as tarefas são retiradas da fila de acordo com a ordem de submissão, mas são executadas concorrentemente e podem acabar em qualquer ordem.
- ***Serial*** – as tarefas são retiradas da fila de acordo com a filosofia FIFO, mas são executadas de forma não concorrente e podem acabar em qualquer ordem.

  A fila *Main* é criada automaticamente pelo sistema e associada à *main thread* da aplicação [21].

6.4 A linguagem Objective-C

O Objective-C é uma linguagem de programação orientada a objetos, criada por Brad Cox e Tom Love, na década de 80. A linguagem é definida como um conjunto pequeno, mas poderoso, de extensões à linguagem C e pela lógica de transmissão de mensagens do *Smalltalk*, uma das primeiras linguagens de programação orientada a objetos.

Presentemente, a linguagem Objective-C é utilizada no desenvolvimento de aplicações para os sistemas operativos *Mac OS X* e *iOS*. Devido ao sucesso e popularização dos dispositivos *iOS*, o Objective-C tem ganho muitos adeptos, tornando-se numas das linguagens mais utilizadas nos últimos tempos.

Como o Objective-C é baseado na linguagem C, é possível compilar qualquer programa em C com um compilador Objective-C. A sua sintaxe deriva tanto da linguagem C como do *Smalltalk*. Especificando, a maior parte da sua sintaxe (expressões, declarações e chamadas de funções) foi herdada da linguagem C, enquanto a vertente de orientação a objetos foi inteiramente definida pela troca de mensagens do *Smalltalk* [8].

Capítulo 7

Conclusões e Trabalho Futuro

A tecnologia móvel, adotada e utilizada em diversos contextos, é caraterizada pela sua globalização e integração no quotidiano. Atingiu uma dimensão simbólica tal, que se tornou necessária e imprescindível. A mudança social é muitas vezes marcada por feitos tecnológicos, como os *tablets* e os *smartphones*, que, de certa forma, traduzem a mudança da sociedade atual, o modo de pensar, o modo de agir e o modo de socializar.

Foi a motivação de satisfazer necessidades e de seguir a tendência evolutiva da tecnologia que levou, a quem negoceia na bolsa, à procura de ter sempre à sua disposição uma plataforma cada vez mais rápida e intuitiva. A corrida e concorrência às aplicações móveis já começaram, e apesar da grande maioria das aplicações disponíveis no mercado apresentarem diversas limitações, quem conseguir vingar neste amplo mercado e apresentar soluções credíveis e seguras, com certeza que terá um futuro risonho. Este desafio que me foi lançado permitiu-me entrar nesta corrida e encarar qualquer projeto futuro com tranquilidade.

Foram vários os objetivos propostos, desde o acesso à informação detalhada dos títulos dos muitos mercados e praças, detalhe de cada título com gráficos de histórico e *intraday*, livro de ordens com todas as funcionalidades de modificação, anulação e duplicação de uma ordem, carteira de títulos e contas bancárias dos clientes e, por fim, a possibilidade do envio de ordens de forma rápida e eficiente. Todos os objetivos foram cumpridos de uma forma gradual ao longo do projeto e as expectativas, quer por parte da empresa acolhedora, quer por parte dos clientes, foram superadas. Portanto, a aplicação teve uma boa aceitação por parte do público.

Para que a concretização destes objetivos fosse possível, foram idealizados e estudados vários padrões de desenho e arquiteturas com o auxílio de protótipos e testes que foram

realizados numa fase inicial. Um trabalho árduo, mas que enriqueceu e fortaleceu os meus conhecimentos. À medida que cada funcionalidade foi sendo implementada e desenhada, foram feitas demonstrações pelos comerciais da Finantech aos clientes. As sugestões foram surgindo, quer por parte dos clientes quer por parte dos próprios colaboradores da Finantech, mais a nível do desenho da interface e navegação da aplicação. A aplicação foi evoluindo a bom ritmo, com os clientes a fornecerem um bom feedback.

O interesse dos clientes pela aplicação iPhone fez surgir a hipótese de a desenvolver para o iPad. Esta permitiria aos clientes, no âmbito empresarial, como corretoras e bancos, terem uma visão mais alargada e próxima de um computador convencional, a que os traders estão normalmente habituados.

Após a concretização de uma versão estável da aplicação, quer para iPhone quer para iPad, a aplicação entrou em fase de testes no Banco Invest e em produção na corretora Fincor.

Para continuar no caminho das aplicações móveis, o próximo passo, como trabalho futuro, e como resposta aos pedidos já existentes dos clientes utilizadores, é a migração das aplicações já desenvolvidas para o sistema *iOS*, para o sistema *Android* e para outras plataformas móveis que estejam de acordo com os padrões delineados.

Apêndice A

Acrónimos

iOS iPhone Operating System

VWAP Volume Weighted Average Price

BCP Banco Comercial Português

PSI20 Portuguese Stock Index

PIN Provider Identification Number

MVC Model View Controller

XNAS NASDAQ

XLIS Euronext Lisbon

CS Common Stock

EDP Energias De Portugal

APPL Apple

TCP Transmission Control Protocol

IP Internet Protocol

GL Gatignol Laurent

MDF Market Data Feed

OBF OrderBook Feed

PF Portfolio Feed

BF Balance Feed

NOF New Order Feed

COF Cancel Order Feed

ROF Replace Order Feed

NF News Feed

CIF Client Identifier Feed

GCD Grand Central Dispatch

API Application Programming Interface

FB Facebook

LLVM Low Level Virtual Machine

SDK Software Development Kit

Mac OS X Macintosh Operating System X

GCC GNU Compiler Colletion

GNU GNU's Not Unix

CPU Central Processesing Unit

UIKit User Interface Kit

2D 2 Dimensions

FIFO First In First Out

HTTP Hypertext Transfer Protocol

P2P Peer-To-Peer

HTTPS Hypertext Transfer Protocol Secure

IIS Internet Information Services

FTP File Transfer Protocol

FTPS File Transfer Protocol SFTP

SFTP Secured File Transfer Protocol

SMTP Simple Mail Transfer Protocol

NNTP Network News Transfer Protocol

SSL Secure Sockets Layer

TLS Transport Layer Security

TCP Transmission Control Protocol

UDP User Datagram Protocol

OSI Open Systems Interconnection

IP Internet Protocol

AES Advanced Encryption Standard

RC4 Ron's Code 4

SHA-1 Secure Hashing Algorithm 1

Referências

[1] Bryan Andrews, Suzanna; Burrough. 50 leaders in the information technology. October 1995.

[2] Alexander Christopher; Sara Ishikawa; Murray Silverstein; Max Jacobson; Ingrid Fiksdahl-King; Shlomo Angel. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1st edition, 1977.

[3] Millennium BCP, setembro 2012. `http://corp.millenniumbcp.pt/pt/public/popuppages/pages/avisolegal.aspx`.

[4] Millennium BCP, setembro 2012. `http://mil.millenniumbcp.pt/site/conteudos/02/article.jhtml?articleID=685675`.

[5] R. Fielding; UC Irvine; J. Gettys; Compaq/W3C; J. Mogul; Compaq; H. Frystyk; W3C/MIT; L. Masinter; Xerox; P. Leach; Microsoft; T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, Internet Engineering Task Force, 1999.

[6] Michael Bloomberg, setembro 2012. `http://www.mikebloomberg.com/index.cfm?objectid=E689D66F-96FD-E9F6-B1AF64B8DAE78A69`.

[7] James O. Coplien. Software design patterns: Common questions and answers. pages 1–2.

[8] Brad Cox. *The object oriented pre-compiler: programming Smalltalk 80 methods in C language*. ACM SIGPLAN Notices, 1nd edition, 1983.

[9] GoBulling, setembro 2012. `http://pro.gobulling.com/sobre-nos`.

[10] GoBulling, setembro 2012. `http://pro.gobulling.com/gobulling-pro/gobulling-pro-mobile`.

[11] GoBulling, setembro 2012. `http://pro.gobulling.com/informacao-legal/informacao-legal`.

[12] P. Gutmann. Using Message Authentication Code (MAC) Encryption in the Cryptographic Message Syntax (CMS). RFC 6476, Internet Engineering Task Force, 2012.

[13] Glenn E.; Stephen T. Pope Krausner. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. pages 1–4, 1988.

[14] J.F. Kurose and K.W. Ross. *Computer Networking: A Top-Down Approach*. Pearson/Addison Wesley, 2008.

[15] Chris Lattner, setembro 2012. `http://llvm.org/Users.html`.

[16] BLOOMBERG L.P, setembro 2012. `http://www.bloomberg.com/company/#history`.

[17] BLOOMBERG L.P, setembro 2012. `http://www.bloomberg.com/mobile/bloomberg/`.

[18] E. Rescorla. HTTP Over TLS. RFC 2818, Internet Engineering Task Force, May 2000.

[19] T. Dierks; Independent; E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, Internet Engineering Task Force, 2008.

[20] Ph.D. Steve Burbeck. Applications programming in smalltalk-80(tm):how to use model-view-controller (mvc). 1987, 1992.

[21] The GCD team, setembro 2012. `http://libdispatch.macosforge.org/`.

[22] Erich Gamma; Richard Helm; Ralph Johnson; John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1st edition, November 10, 1994.